NORTHWESTERN UNIVERSITY

Efficient Second-Order Methods for Second-Order Cone Programs and Continuous Nonlinear Two-Stage Optimization Problems

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Xinyi Luo

EVANSTON, ILLINOIS

September 2023

Abstract

Efficient Second-Order Methods for Second-Order Cone Programs and Continuous Nonlinear Two-Stage Optimization Problems

Xinyi Luo

This dissertation presents novel advancements in the field of continuous nonlinear optimization, focusing on the development of efficient second-order methods for second-order conic programs (SOCPs) and continuous nonlinear two-stage optimization problems. The primary focus is on the theory and computations of Sequential Quadratic Programming (SQP) methods, which are widely used for solving nonlinear optimization problems.

In this work, we introduce a new SQP method tailored for linear SOCPs, leveraging the SQP framework for nonlinear optimization. By capitalizing on warm-start capabilities for active-set quadratic programming subproblem solvers and utilizing polyhedral outer approximations of cones, our method achieves a local quadratic rate of convergence. It efficiently identifies the set of cones where the optimal solution lies at extreme points, enabling rapid local convergence. Numerical experiments confirm that the method can take advantage of good starting points and can achieve higher accuracy compared to a state-of-the-art interior point solver.

The dissertation also includes two computational projects. Firstly, we introduce RestartSQP, an open-source C++ software package implementing Fletcher's Sℓ_1QP method and integrating the parametric active-set methods for solving SQP subproblems. RestartSQP offers unique warm-start capabilities, efficiently handling changing constraint values and incorporating additional constraints dynamically. Additionally, it supports crossover from interior-point solvers like Ipopt, enabling the solution of subsequent NLPs using the SQP method. Numerical experiments using the CUTE benchmark NLP test set demonstrate the practical performance of RestartSQP.

Furthermore, we develop a two-stage decomposition algorithm for continuous nonlinear two-stage optimization problems and discussed the detailed C++ implementation. The algorithm decomposes large-scale problems into master and independent subproblems, allowing for parallel processing and accelerated speedup. To address non-differentiability challenges arising in subproblem objectives, we incorporate a smoothing technique and propose a new approximation scheme for subproblem objectives. Furthermore, an extrapolation strategy is utilized to enhance the computational efficiency of the algorithm. By combining RestartSQP and the interior-point solver Ipopt, our algorithm demonstrates parallel scalability and efficient solution to large-scale optimization problems.

Acknowledgements

I am deeply indebted to my supervisor Professor Andreas Wächter, whose unwavering support and expertise have been invaluable throughout this journey. Andreas's insightful feedback, patience, and dedication have significantly improved the quality of this work. His high standards and encouragement have constantly motivated me to strive for excellence. I am truly thankful for his guidance on my path of numerical optimization, which has been invaluable to my growth as a researcher.

I would like to express my gratitude to Professor Ermin Wei and Dr. Russell Bent for their generous contribution as members of my prospectus and thesis defense committee. Their insightful feedback has been invaluable in refining my academic work. I am grateful for their time, expertise, and commitment to helping me succeed in my research endeavors.

I am deeply grateful to all the individuals at Los Alamos National Laboratory's Advanced Network Science Initiative for their invaluable contributions to my academic journey. My summer internship in New Mexico was truly memorable, and I owe a debt of gratitude to several people who made it a rewarding experience. I extend my sincere thanks to Carleton and Hassan for guiding my internship throughout the summer. I would like to express my heartfelt appreciation to David, Russell, and Juan for their assistance and collaboration in the development of the two-stage optimization algorithm project.

My heartfelt thanks extend to my friends and colleagues who provided unwavering emotional support, lifting my spirits over the course of the five-year Ph.D. journey. I am grateful

to the entire Optimization Lab L375, including the past members Ruby, Yuchen, Alejandra, and Michael, and the current members Shima, Shigeng, Melody, and Xiaochun. I would also like to express my gratitude to my friends in the IEMS department, including Kim, Tong, Yufeng, and Jeffery. Your friendship and companionship have made Northwestern feel like a second home to me.

To my parents, thank you for your endless love, unwavering support, and strong belief in me. Without you, I would not be who I am today.

Above all, I would like to thank my fiancé Boyi for his love and constant support. Thank you for being my inspiration, my closest friend, and the one I love.

To all those mentioned, and even those whose names might not appear here, I sincerely thank you for being part of my journey. This journey has been challenging, yet immensely rewarding, and I am honored to have had such incredible support along the way.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Sequential Quadratic Programming (SQP) is a method for solving Nonlinear Programming (NLP) problems. These problems are typically formulated as follows:

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

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

$$d(x) \leq 0. \tag{1.1c}$$

In this formulation, the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the constraint functions $c : \mathbb{R}^n \to \mathbb{R}^{m_c}$ and $d : \mathbb{R}^n \to \mathbb{R}^{m_d}$ are all nonlinear and are assumed to be twice continuously differentiable.

SQP methods have been extensively studied in the field of nonlinear programming [**33, 29, 28**]. It is an iterative algorithm that utilizes Quadratic Programming (QP) to model (1.1) at a given iterate x^k. At an iterate x^k, the basic SQP method computes a step p^k as an optimal solution to the QP subproblem

$$\begin{aligned} &\min_{p \in \mathbb{R}^n} \nabla f(x^k)^T p + \tfrac{1}{2} p^T H^k p \\ &\text{s.t. } c(x^k) + \nabla c(x^k)^T p = 0, \\ &\quad d(x^k) + \nabla d(x^k)^T p \leq 0. \end{aligned}$$

where H^k is a symmetric matrix that is constructed using the second-order information of f, c and d, $\nabla f(x)$ is the gradient of the objective function $f(x)$ and $\nabla c(x)^T = (\nabla c_1(x), \cdots, \nabla c_{m_c}(x))^T$ is the Jacobian of c at x. $\nabla d(x)^T$ is defined analogously. The solution of the QP subproblem provides the search direction p^k, which is then used to construct a new iterate x^{k+1}. The SQP method constructs a sequence of iterates $\{(x^k, \lambda^k, \mu^k)\}$ whose limit points are KKT points of (1.1), where λ^k and μ^k are the multiplier estimates of (1.1b) and (1.1c) at k-th iteration.

SQP methods can be viewed as a natural extension of Newton's method for constrained optimization problems. Notably, under mild regularity conditions, these methods exhibit similar quadratic rate convergence properties to Newton's method when iterates are close to an optimal solution (x^*, λ^*, μ^*). This rapid convergence is a significant strength of SQP methods and makes them appealing for solving a wide range of optimization problems beyond regular NLPs.

In this dissertation, we have extended the SQP framework to create efficient second-order methods for a broader set of problems. These methods are specifically designed to handle two types of optimization problems: second-order conic programs (SOCPs) and continuous nonlinear two-stage optimization problems.

1.1. Sequential Quadratic Programming Method for Second-Order Conic Programs

In Chapter 2, we propose a new method for linear second-order cone programs of the following form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \; & c^T x \\ \text{s.t. } & Ax \leq b, \\ & x_j \in \mathcal{K}_j \qquad j \in \mathcal{J} := \{1, \ldots, p\}, \end{aligned}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and x_j is a subvector of x of dimension n_j with index set $\mathcal{I}_j \subseteq \{1, \ldots, n\}$. We assume that the sets $\mathcal{I}_j$ are disjoint. The set $\mathcal{K}_j$ is the second-order cone of dimension n_j.

The method is based on the sequential quadratic programming framework for nonlinear programming. In contrast to interior point methods, it can capitalize on the warm-start capabilities of active-set quadratic programming subproblem solvers and achieve a local quadratic rate of convergence.

To address the non-differentiability challenges posed by the nonlinear formulations of conic constraints, the subproblems approximate the cones using polyhedral outer approximations, which are continuously refined during the iterations. In cases of nondegenerate instances, the algorithm implicitly identifies the set of cones for which the optimal solution lies at the extreme points. Consequently, the final steps are identical to regular sequential quadratic programming steps for differentiable nonlinear optimization problems, resulting in local quadratic convergence.

1.2. Nonlinear Two-Stage Decomposition Algorithm

Another topic addressed in this dissertation is a decomposition algorithm for continuous nonlinear two-stage optimization problems. The algorithm is designed to solve the following problem:

$$\begin{aligned} \min_{x_i \in \mathbb{R}^{n_i^P}, z \in \mathbb{R}^{n_0}} \quad & f_0(z) + \sum_{i=1}^{N} \hat{f}_i(x_i) \\ \text{s.t } & c_0(z) = 0, \\ & d_0(z) \leq 0, \\ & \hat{P}_i z - x_i = 0, \quad i = 1, \ldots, N, \end{aligned}$$

together with N ($N \geq 0$) second-stage problems of the form

$$\begin{aligned} \hat{f}_i(x_i) = \min_{y_i \in \mathbb{R}^{n_i}} \quad & f_i(y_i) \\ \text{s.t } & c_i(y_i) = 0, \\ & d_i(y_i) \leq 0, \\ & P_i y_i - x_i = 0. \end{aligned}$$

Here, $f_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}$, $c_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}^{m_i^c}$, and $d_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}^{m_i^d}$ are assumed to be twice-continuously differentiable. The shared variables $x_i \in \mathbb{R}^{n_i^P}$ are subvectors of the first-stage variables z defined using projection matrix $\hat{P}_i \in \mathbb{R}^{n_i^P \times n_0}$. Similarly, the second-stage variables y_i is defined by the projection matrix $P_i \in \mathbb{R}^{n_i^P \times n_i}$.

Chapter 4 presents a comprehensive description of the decomposition algorithm. The proposed approach decomposes the original problem into a master problem and a set of independent subproblems. This decomposition enables the utilization of parallel processing, leading to accelerated speedup and enhanced efficiency.

A key strength of this new algorithm lies in its ability to employ second-order optimization solvers to handle the nonlinearity of the instances in a natural manner. By breaking down the problem into manageable subproblems, each solver can efficiently tackle its corresponding part, thus collectively contributing to the solution of the entire large-scale problem.

To make this work, we address a challenge that makes the direct application of a fast second-order nonlinear optimization solver for the first-stage problems impossible: the second-stage value function $\hat{f}_i$ is typically not differentiable at values of x_i when the active-set changes. To overcome this difficulty, the proposed algorithm relies on a barrier-problem reformulation that results in a smooth approximation of the value function $\hat{f}_i$ so that a standard nonlinear optimization method can be used to solve the smoothed first-stage problem.

The implementation of the two-stage decomposition algorithm is realized in C++ and includes the development of the RestartSQP solver, an open-source software package based on Fletcher's Sℓ_1QP method [**27**]. RestartSQP is integrated with the parametric active-set method for solving SQP subproblems, providing warm-start capabilities and efficient handling of changing constraint values and dynamic incorporation of additional constraints. The software also supports crossover from interior-point solvers like Ipopt, facilitating the solution of subsequent NLPs using the SQP method.

Chapter 3 of the dissertation presents detailed insights into the implementation of RestartSQP and provides numerical results using the CUTE benchmark NLP test set [**12**]. These results demonstrate the practical performance of RestartSQP.

In the two-stage decomposition algorithm, RestartSQP plays a crucial role as the master problem solver. Due to that SQP methods require fewer function evaluations compared to interior-point methods, RestartSQP is well-suited for solving the master problem in the two-stage decomposition algorithm. On the other hand, the interior-point solver Ipopt is utilized to handle the subproblems. Additionally, the software provides interfaces for modeling languages including AMLP and JuMP. Chapter 4 provides numerical results showcasing the parallel scalability of the two-stage decomposition algorithm, confirming its effectiveness in solving large-scale problems efficiently.

1.3. Notation and Definitions

Throughout the dissertation, the following notations and definitions will be used.

We use $[n]$ to denote the index set $\{1, \cdots, n\}$. For two vectors $x, y \in \mathbb{R}^n$, we denote with $x \circ y$ their component-wise product, and the condition $x \perp y$ stands for $x^T y = 0$. For $x \in \mathbb{R}^n$, we define $[x]^+$ as the vector with entries $\max\{x_i, 0\}$. We denote by $\|\cdot\|$, $\|\cdot\|_1$, $\|\cdot\|_\infty$ the Euclidean norm, the ℓ_1-norm, and the ℓ_∞-norm, respectively. If a function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, we denote its gradient by $\nabla_x f(x)$. If f is twice-differentiable, we denote its Hessian by $\nabla^2_{xx} f(x)$. For a cone $\mathcal{K}_j$, $e_{ji} \in \mathbb{R}^{n_j}$ is the canonical basis vector with 1 in the element corresponding to x_{ji} for $i \in \{0, \ldots, n_j - 1\}$, and $\text{int}(\mathcal{K}_j)$ and $\text{bd}(\mathcal{K}_j)$ denote the cone's interior and boundary, respectively.

Definition 1.3.1 (Lagrangian function)**.** *We define the Lagrangian function associated with the NLP* (1.1) *as*

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T c(x) + \mu^T d(x).$$

The vector $\lambda \in \mathbb{R}^{m_c}$ and $\mu \in \mathbb{R}^{m_d}$ are referred to as Lagragian multipliers.

Definition 1.3.2 (Active set)**.** *For $x \in \mathbb{R}^n$, the index set*

$$\mathcal{A}(x) := \{1, \cdots, m_c\} \cup \{i \in \{1, \cdots, m_d\} | d_i(x) = 0\}$$

is referred to as the active set at x.

Definition 1.3.3 (Linear independence constraint qualification)**.** *Given the point x and the active set $\mathcal{A}(x)$, we say that the linear independence constraint qualification (LICQ) holds if the gradients of the active constraints $\{\nabla c_i(x), i \in [m_c]\} \cup \{\nabla d_i(x), i \in \mathcal{A}(x)\}$ are linearly independent.*

Definition 1.3.4 (First-order necessary optimality conditions)**.** *Suppose x^* is a local solution of* (1.1) *and that LICQ holds at x^*. Then there are Lagrange multiplier vectors λ^* and μ^* such that the following conditions are satisfied at (x^*, λ^*, μ^*)*

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0,$$

$$c(x^*) = 0, \qquad d(x^*) \leq 0,$$

$$\mu^* \geq 0, \quad \mu^* \circ d(x^*) = 0.$$

These conditions are also referred to as Karush-Kuhn-Tucker(KKT) conditions.

Definition 1.3.5 (Strict Complementarity)**.** *Given a local solution x^* of* (1.1) *and vectors λ^* and μ^* satisfying* (1.2)*, we say the strict complementarity condition holds if $\mu_i^* > 0$ for each $i \in \mathcal{A}(x^*)$.*

Definition 1.3.6 (Critical cone)**.** *Given a local solution x^* of* (1.1) *and vectors λ^* and μ^* satisfying* (1.2)*, we define the critical cone at x^* as*

$$\mathcal{C}(x^*, \lambda^*, \mu^*) = \left\{ d \in \mathbb{R}^n \middle| \begin{array}{l} \nabla c_i(x^*)^T d = 0, \forall i \in [m_c], \\ \nabla d_i(x^*)^T d = 0, \forall i \in [m_d] \cap \mathcal{A}(x^*) \text{ with } \mu_i^* > 0, \\ \nabla d_i(x^*)^T d \leq 0, \forall i \in [m_d] \cap \mathcal{A}(x^*) \text{ with } \mu_i^* = 0 \end{array} \right\}.$$

Definition 1.3.7 (Second-order sufficient optimality conditions)**.** *Suppose that for some feasible $x \in \mathbb{R}^n$, there are vectors $\lambda \in \mathbb{R}^{m_c}$ and $\mu \in \mathbb{R}^{m_d}$ such that the KKT conditions* (1.2) *are satisfied. Suppose also that*

$$d^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*, \mu^*) d > 0, \quad \forall d \in \mathcal{C}(x^*, \lambda^*, \mu^*) \setminus \{0\}.$$

Then x^* is a strict local minimizer of (1.1)*.*

CHAPTER 2

A Quadratically Convergent Sequential Programming Method for Second-Order Cone Programs Capable of Warm Starts

2.1. Introduction

We are interested in the solution of second-order cone programs (SOCPs) of the form

$$\min_{x \in \mathbb{R}^n} \; c^T x \tag{2.1a}$$

$$\text{s.t. } Ax \leq b, \tag{2.1b}$$

$$x_j \in \mathcal{K}_j \qquad j \in \mathcal{J} := \{1, \ldots, p\}, \tag{2.1c}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and x_j is a subvector of x of dimension n_j with index set $\mathcal{I}_j \subseteq \{1, \ldots, n\}$. We assume that the sets $\mathcal{I}_j$ are disjoint. The set $\mathcal{K}_j$ is the second-order cone of dimension n_j, i.e.,

$$\mathcal{K}_j := \{y \in \mathbb{R}^{n_j} : \|\bar{y}\| \leq y_0\}, \tag{2.2}$$

where the vector y is partitioned into $y = (y_0, \bar{y}^T)^T$ with $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_{n_j - 1})^T$. These problems arise in a number of important applications [**2, 5, 57, 75**]

Currently, most of the commercial software for solving SOCPs implements interior-point algorithms which utilize a barrier function for second-order cones, see, e.g. [**41, 47, 3**]. Interior-point methods have well-established global and local convergence guarantees [**63**] and are able to solve large-scale instances, but they cannot take as much of an advantage

of a good estimate of the optimal solution as it would be desirable in many situations. For example, in certain applications, such as online optimal control, the same optimization problem has to be solved over and over again, with slightly modified data. In such a case, the optimal solution of one problem provides a good approximation for the new instance. Having a solver that is capable of "warm-starts", i.e., utilizing this knowledge, can be essential when many similar problems have to be solved in a small amount of time.

For some problem classes, including linear programs (LPs), quadratic programs (QPs), or nonlinear programming (NLP), active-set methods offer suitable alternatives to interior-point methods. They explicitly identify the set of constraints that are active (binding) at the optimal solution. When these methods are started from a guess of the active set that is close to the optimal one, they often converge rapidly in a small number of iterations. An example of this is the simplex method for LPs. Its warm-start capabilities are indispensable for efficient branch-and-bound algorithms for mixed-integer linear programs.

Active-set methods for LPs, QPs, or NLPs are also known to outperform interior-point algorithms for problems that are not too large [**38, 45, 51**]. Similarly, active-set methods might be preferable when there are a large number of inequality constraints among which only a few are active, since an interior-point method is designed to consider all inequality constraints in every iteration and consequently solves large linear systems, whereas an active set method can ignore all inactive inequality constraints and encounters potentially much smaller linear systems.

Our goal is to propose an active-set alternative to the interior-point method in the context of SOCP that might provide similar benefits. We introduce a new sequential quadratic programming (SQP) algorithm that, in contrast to interior-point algorithms for SOCPs, has favorable warm-starting capabilities because it can utilize active-set QP solvers. We

prove that it is globally convergent, i.e., all limit points of the generated iterates are optimal solutions under mild assumptions, and that it enjoys a quadratic convergence rate for non-degenerate instances. Our preliminary numerical experiments demonstrate that these theoretical properties are indeed observed in practice. They also show that the algorithm is able in some cases to compute a solution to a higher degree of precision than interior point methods. This is expected, again in analogy to the context of LPs, QPs, and NLPs, since an interior point method terminates at a small, but nonzero value of the barrier parameter that cannot be made smaller than some threshold (typically 10^{-6} or 10^{-8}) because the arising linear systems become highly ill-conditioned. In contrast, in the final iteration of the active-set method, the linear systems solved correspond directly to the optimality conditions, without any perturbation introduced by a barrier parameter, and are only as degenerate as the optimal solution of the problem.

The paper is structured as follows. Chapter 2.2 reviews the sequential quadratic programming method and the optimality conditions of SOCPs. Chapter 2.3 describes the algorithm, which is based on an outer approximation of the conic constraints. Chapter 2.4 establishes the global and local convergence properties of the method, and numerical experiments are reported in Chapter 4.7. Concluding remarks are offered in Chapter 2.6.

2.1.1. Related work

While a large number of interior-point algorithms for SOCP have been proposed, including some that have been implemented in efficient optimization packages [**41, 47, 3**], there are only very few approaches for solving SOCPs with an active-set framework. The method proposed by Goldberg and Leyffer [**35**] is a two-phase algorithm that combines a projected-gradient method with equality-constrained SQP. However, it is limited to instances that have

only conic constraints (2.1c) and no additional linear constraints (2.1b). Hayashi et al. [**44**] propose a simplex-type method, where they reformulate the SOCP as a linear semi-infinite program to handle the fact that these instances have infinitely many extreme points. The resulting dual-simplex exchange method shows promising practical behavior. However, in contrast to the method proposed here, the authors conjecture that their method has only an R-linear local convergence rate. Zhadan [**86**] proposes a similar simplex-type method. Another advantage of the method presented in this paper is that the pivoting algorithm does not need to be designed and implemented from scratch. Instead, it can leverage existing implementations of active-set QP solvers, in particular the efficient handling of linear systems.

The proposed algorithm relies on polyhedral outer approximations based on well-known cutting planes for SOCPs. For instance, the methods for mixed-integer SOCP by Drewes and Ulbrich [**23**] and Coey et al. [**17**] use these cutting planes to build LP relaxations of the branch-and-bound subproblems. We note that an LP-based cutting plane algorithm for SOCP could be seen as an active-set method, but it is only linearly convergent. As pointed out in [**21**], it is crucial to consider the curvature of the conic constraint in the subproblem objective to achieve fast convergence.

The term "SQP method for SOCP" has also been used in the literature to refer to methods for solving nonlinear SOCPs [**21, 49, 60, 87**]. However, in contrast to the method here, in these approaches, the subproblems themselves are SOCPs (2.1) and include the linearization of the nonlinear objective and constraints. It will be interesting to explore extensions of the proposed method to nonlinear SOCPs in which feasibility is achieved asymptotically not only for the nonlinear constraints but also for the conic constraints.

2.2. Preliminaries

The NLP reformulation of the SOCP is introduced in Chapter 2.2.1. We review in Chapter 2.2.2 the local convergence properties of the SQP method and in Chapter 2.2.3 the penalty function as a means to promote convergence from any starting point. In Chapter 2.2.4, we briefly state the optimality conditions and our assumptions for the SOCP (2.1).

2.2.1. Reformulation as a smooth optimization problem

The definition of the second-order cone in (2.2) suggests that the conic constraint (2.1c) can be replaced by the nonlinear constraint

$$r_j(x_j) := \|\bar{x}_j\| - x_{j0} \leq 0$$

without changing the set of feasible points. Consequently, (2.1) is equivalent to

$$\min_{x \in \mathbb{R}^n} \; c^T x \tag{2.3a}$$

$$\text{s.t. } Ax \leq b, \tag{2.3b}$$

$$r_j(x_j) \leq 0, \qquad j \in \mathcal{J}. \tag{2.3c}$$

Unfortunately, (2.3) cannot be solved directly with standard gradient-based algorithms for nonlinear optimization, such as SQP methods. The reason is that r_j is not differentiable whenever $\bar{x}_j = 0$. This is particularly problematic when the optimal solution x^* of the SOCP lies at the extreme point of a cone, $x_j^* = 0 \in \mathcal{K}_j$. In that case, the Karush-Kuhn-Tucker (KKT) necessary optimality conditions for the NLP formulation, which are expressed in terms of derivatives, cannot be satisfied. Therefore, any optimization algorithm that seeks

KKT points cannot succeed. As a remedy, differentiable approximations of r_j have been proposed in the past; see, for example, [**80**]. However, high accuracy comes at the price of high curvature, which can make finding the numerical solution of the NLP difficult.

An alternative equivalent reformulation of the conic constraint is given by

$$\|\bar{x}_j\|^2 - x_{j0}^2 \leq 0 \text{ and } x_{j0} \geq 0.$$

In this case, the constraint function is differentiable. But if $x_j^* = 0$, its gradient vanishes, and as a consequence, no constraint qualification applies and the KKT conditions do not hold. Therefore, again, a gradient-based method cannot be employed. By using an outer approximation of the cones that is improved in the course of the algorithm, our proposed variation of the SQP method is able to avoid these kinds of degeneracy.

To facilitate the discussion we define a point-wise partition of the cones.

Definition 2.2.1. *Let $x \in \mathbb{R}^n$.*

(1) *We call a cone $\mathcal{K}_j$* extremal-active *at x, if $x_j = 0$, and we denote with $\mathcal{E}(x) = \{j \in \mathcal{J} : x_j = 0\}$ the set of extremal-active cones at x.*

(2) *We define the set $\mathcal{D}(x) = \{j \in \mathcal{J} : \bar{x}_j \neq 0\}$ as the set of all cones for which the function r_j is differentiable at x.*

(3) *We define the set $\mathcal{N}(x) = \{j \in \mathcal{J} : x_j \neq 0$ and $\bar{x}_j = 0\}$ as the set of all cones that are not extremal-active and for which r_j is not differentiable x.*

If the set $\mathcal{E}(x^*)$ at an optimal solution x^* were known in advance, we could compute x^* as a solution of (2.1) by solving the NLP

$$\min_{x \in \mathbb{R}^n} c^T x \tag{2.4a}$$

$$\text{s.t. } Ax \leq b, \tag{2.4b}$$

$$r_j(x) \leq 0, \quad j \in \mathcal{D}(x^*), \tag{2.4c}$$

$$x_j = 0, \qquad j \in \mathcal{E}(x^*). \tag{2.4d}$$

The constraints involving the linearization of r_j are imposed only if r_j is differentiable at x^*, and variables in cones that are extremal-active at x^* are explicitly fixed to zero. With this, locally around x^*, all functions in (2.4) are differentiable and we could apply standard second-order algorithms to achieve fast local convergence.

In (2.4), we omitted the cones in $\mathcal{N}(x^*)$. If x^* is feasible for the SOCP and $j \in \mathcal{N}(x^*)$ we have $\bar{x}_j^* = 0$ and $x_{j0}^* > 0$, and so $r_j(x^*) < 0$. This implies that the nonlinear constraint (2.4c) for this cone is not active and we can omit it from the problem statement without impacting the optimal solution.

2.2.2. Local convergence of SQP methods

The proposed algorithm is designed to guide the iterates x^k into the neighborhood of an optimal solution x^*. If the optimal solution is not degenerate and the iterates are sufficiently close to x^*, the steps generated by the algorithm are eventually identical to the steps that the SQP method would take for solving the differentiable optimization problem (2.4). In this chapter, we review the mechanisms and convergence results of the basic SQP method [**59**].

At an iterate x^k, the basic SQP method, applied to (2.4), computes a step d^k as an optimal solution to the QP subproblem

$$\min_{d\in\mathbb{R}^n} \; c^T d + \tfrac{1}{2} d^T H^k d \tag{2.5a}$$

$$\text{s.t. } A(x^k + d) \le b, \tag{2.5b}$$

$$r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j \le 0, \qquad j \in \mathcal{D}(x^*), \tag{2.5c}$$

$$x_j^k + d_j = 0, \qquad j \in \mathcal{E}(x^*). \tag{2.5d}$$

Here, H^k is the Hessian of the Lagrangian function for (2.4), which in our case is

$$H^k = \sum_{j\in\mathcal{D}(x^*)} \mu_j^k \nabla_{xx}^2 r_j(x_j^k), \tag{2.6}$$

where $\mu_j^k \ge 0$ are estimates of the optimal multipliers for the nonlinear constraint (2.4c), and where $\nabla_{xx}^2 r_j(x_j)$ is the $n \times n$ block-diagonal matrix with

$$\nabla^2 r_j(x_j) = \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{\|\bar{x}_j\|} I - \frac{\bar{x}_j \bar{x}_j^T}{\|\bar{x}_j\|^3} \end{bmatrix} \tag{2.7}$$

in the rows and columns corresponding to x_j for $j \in \mathcal{J}$. It is easy to see that $\nabla^2 r_j(x_j)$ is positive semi-definite. The estimates μ_j^k are updated based on the optimal multipliers $\hat{\mu}_j^k \ge 0$ corresponding to (2.5c).

Algorithm 1 Basic SQP Algorithm

Require: Initial iterate x^0 and multiplier estimates λ^0, μ^0, and η^0.

1: **for** $k = 0, 1, 2 \cdots$ **do**
2: Compute H^k from (2.6).
3: Solve QP (2.5) to get step d^k and multipliers $\hat{\lambda}^k$, $\hat{\mu}_j^k$, and $\hat{\eta}^k$.
4: Set $x^{k+1} \leftarrow x^k + d^k$ and $\mu_j^{k+1} \leftarrow \hat{\mu}_j^k$ for all $j \in \mathcal{D}(x^*)$.
5: **end for**

Algorithm 1 formally states the basic SQP method where $\hat{\lambda}^k \geq 0$ and $\hat{\eta}^k$ denote the multipliers corresponding to (2.4b) and (2.4d), respectively. Because we are only interested in the behavior of the algorithm when x^k is close to x^*, we assume here that $\bar{x}_j^k \neq 0$ for all $j \in \mathcal{D}(x^*)$ and for all k, and hence the gradient and Hessian of r_j can be computed. Note that the iterates $\hat{\lambda}^k$ and $\hat{\eta}^k$ are not explicitly needed in Algorithm 1, but they are necessary to measure the optimality error and define the primal-dual iterate sequence that is analyzed in Theorem 2.2.1.

A fast rate of convergence can be proven under the following sufficient second-order optimality assumptions [**11**].

Assumption 2.2.1. *Suppose that x^* is an optimal solution of the NLP (2.4) with corresponding KKT multipliers λ^*, μ^*, and η^*, satisfying the following properties:*

(1) *Strict complementarity holds;*

(2) *the linear independence constraint qualification (LICQ) holds at x^*, i.e., the gradients of the constraints that hold with equality at x^* are linearly independent;*

(3) *the projection of the Lagrangian Hessian $H^* = \sum_{j \in \mathcal{D}(x^*)} \mu_j^* \nabla_{xx}^2 r_j(x_j^*)$ into the null space of the gradients of the active constraints is positive definite.*

Under these assumptions, the basic SQP algorithm reduces to Newton's method applied to the optimality conditions of (2.4) and the following result holds [**59**].

Theorem 2.2.1. *Suppose that Assumption 2.2.1 holds and that the initial iterate x^0 and multipliers μ^0 (used in the Hessian calculation) are sufficiently close to x^* and μ^*, respectively. Then the iterates $(x^{k+1}, \hat{\lambda}^k, \hat{\mu}^k, \hat{\eta}^k)$ generated by the basic SQP algorithm, Algorithm 1, converge to $(x^*, \lambda^*, \mu^*, \eta^*)$ at a quadratic rate.*

2.2.3. Penalty function

Theorem 2.2.1 is a local convergence result. Practical SQP algorithms include mechanisms that make sure that the iterates eventually reach such a neighborhood, even if the starting point is far away. To this end, we employ the exact penalty function

$$\varphi(x;\rho) = c^T x + \rho \sum_{j\in\mathcal{J}} [r_j(x_j)]^+ \tag{2.8}$$

in which $\rho > 0$ is a penalty parameter. Note that we define φ in terms of all conic constraints $\mathcal{J}$, even though r_j appears in (2.4c) only for $j \in \mathcal{D}(x^*)$. We do this because the proposed algorithm does not know $\mathcal{D}(x^*)$ in advance and the violation of all cone constraints needs to be taken into account when the original problem (2.1) is solved. Nevertheless, in this chapter, we may safely ignore the terms for $j \notin \mathcal{D}(x^*)$ because for $j \in \mathcal{E}(x^*)$ we have $x_j^k = 0$ and hence $[r_j(x^k)]^+ = 0$ for all k due to (2.5d), and when $j \in \mathcal{N}(x^*)$, we have $r_j(x_j^k) < 0$ when x^k is close to x^* since $r_j(x_j^*) < 0$.

It can be shown, under suitable assumptions, that the minimizers of $\varphi(\,\cdot\,;\rho)$ over the set defined by the linear constraints (2.4b),

$$X = \{x \in \mathbb{R}^n : Ax \leq b\}, \tag{2.9}$$

coincide with the minimizers of (2.4) when ρ is chosen sufficiently large. Because it is not known upfront how large ρ needs to be, the algorithm uses an estimate, ρ^k, in iteration k, which might be increased during the course of the algorithm.

To ensure that the iterates eventually reach a minimizer of $\varphi(\,\cdot\,;\rho)$, and therefore a solution of (2.4), we require that the decrease of $\varphi(\,\cdot\,;\rho)$ is at least a fraction of that achieved in the piece-wise linear model of $\varphi(\,\cdot\,;\rho)$ given by

$$m^k(x^k+d;\rho) = c^T(x^k+d) + \rho \sum_{j\in\mathcal{D}(x^k)} [r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j]^+, \tag{2.10}$$

constructed at x^k. More precisely, the algorithm accepts a trial point $\hat{x}^{k+1} = x^k + d$ as a new iterate only if the sufficient decrease condition

$$\begin{aligned}\varphi(\hat{x}^{k+1};\rho^k) - \varphi(x^k;\rho^k) &\le c_{\text{dec}}\Big(m^k(x^k+d;\rho^k) - m^k(x^k;\rho^k)\Big) \\ &\overset{(2.10)}{=} c_{\text{dec}}\left(c^T d - \rho^k \sum_{j\in\mathcal{D}(x^k)} [r_j(x_j^k)]^+\right)\end{aligned} \tag{2.11}$$

holds with some fixed constant $c_{\text{dec}} \in (0,1)$. The trial iterate $\hat{x}^{k+1} = x^k + d^k$ with d^k computed from (2.5) might not always satisfy this condition. The proposed algorithm generates a sequence of improved steps of which one is eventually accepted.

However, to apply Theorem 2.2.1, it would be necessary that the algorithm take the original step d^k computed from (2.5); see Step 4 of Algorithm 1. Unfortunately, $\hat{x}^{k+1} = x^k + d^k$ might not be acceptable even when the iterate x^k is arbitrarily close to a non-degenerate solution x^* satisfying Assumption 2.2.1 (a phenomenon called the Maratos effect [**55**]). Our remedy is to employ the second-order correction step [**26**], s^k, which is obtained as an optimal solution of the QP

$$\min_{s\in\mathbb{R}^n} \; c^T(d^k+s) + \tfrac{1}{2}(d^k+s)^T H^k (d^k+s) \tag{2.12a}$$

$$\text{s.t. } A(x^k + d^k + s) \le b, \tag{2.12b}$$

$$r_j(x_j^k + d_j^k) + \nabla r_j(x_j^k + d_j^k)^T s_j \le 0, \qquad j \in \mathcal{D}(x^*), \tag{2.12c}$$

$$x_j^k + d_j^k + s_j = 0, \qquad j \in \mathcal{E}(x^*). \tag{2.12d}$$

For later reference, let $\hat{\lambda}^{S,k}$, $\hat{\mu}^{S,k}$ and $\hat{\eta}^{S,k}$ denote optimal multiplier vectors corresponding to (2.12b)–(2.12d), respectively. The algorithm accepts the trial point $\hat{x}^{k+1} = x^k + d^k + s^k$ if it yields sufficient decrease (2.11) with respect to the original SQP step $d = d^k$. Note that (2.12) is a variation of the second-order correction that is usually used in SQP methods, for which (2.12c) reads

$$r_j(x_j^k + d_j^k) + \nabla r_j(x_j^k)^T(d_j^k + s_j) \leq 0, \qquad j \in \mathcal{D}(x^*),$$

and avoids the evaluation of $\nabla r_j(x_j^k + d_j^k)$. In our setting, however, evaluating $\nabla r_j(x_j^k + d_j^k)$ takes no extra work and (2.12c) is equivalent to a supporting hyperplane, see Chapter 2.3.1. As the following theorem shows (see, e.g., [**26**] or [**18**, Section 15.3.2.3]), this procedure computes steps with sufficient decrease (2.11) and results in quadratic convergence.

Theorem 2.2.2. *Let Assumption 2.2.1 hold and assume that the initial iterate x^0 and multipliers μ^0 are sufficiently close to x^* and μ^*, respectively. Further suppose that $\rho^k = \rho^\infty$ for large k where $\rho^\infty > \mu_j^*$ for all $j \in \mathcal{D}(x^*)$.*

(1) *Consider an algorithm that generates a sequence of iterates by setting $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}, \eta^{k+1}) = (x^k + d^k, \hat{\lambda}^k, \hat{\mu}^k, \hat{\eta}^k)$ or $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}, \eta^{k+1}) = (x^k + d^k + s^k, \hat{\lambda}^{S,k}, \hat{\mu}^{S,k}, \hat{\eta}^{S,k})$ for all $k = 0, 1, 2, \ldots$. Then $(x^k, \lambda^k, \mu^k, \eta^k)$ converges to $(x^*, \lambda^*, \mu^*, \eta^*)$ at a quadratic rate.*

(2) *Further, for all k, either $\hat{x}^{k+1} = x^k + d^k$ or $\hat{x}^{k+1} = x^k + d^k + s^k$ satisfies the acceptance criterion* (2.11).

2.2.4. Optimality conditions for SOCP

The proposed algorithm aims at finding an optimal solution of the SOCP (2.1), or equivalently, values of the primal variables, $x^* \in \mathbb{R}^n$, and the dual variables, $\lambda^* \in \mathbb{R}^m$ and $z_j^* \in \mathbb{R}^{n_j}$ for $j \in \mathcal{J}$, that satisfy the necessary and sufficient optimality conditions [**2**, Theorem 16]

$$c + A^T\lambda^* - z^* = 0, \tag{2.13a}$$

$$Ax^* - b \leq 0 \perp \lambda^* \geq 0, \tag{2.13b}$$

$$\mathcal{K}_j \ni x_j^* \perp z_j^* \in \mathcal{K}_j, \qquad j \in \mathcal{J}. \tag{2.13c}$$

A thorough discussion of SOCPs is given in the comprehensive review by Alizadeh and Goldfarb [**2**]. The authors consider the formulation in which the linear constraints (2.1b) are equality constraints, but the results in [**2**] can be easily extended to inequalities.

The primal-dual solution (x^*, λ^*, z^*) is unique under the following assumption.

Assumption 2.2.2. *(x^*, λ^*, z^*) is a non-degenerate primal-dual solution of the SOCP (2.1) at which strict complementarity holds.*

The definition of non-degeneracy for SOCP is somewhat involved and we refer the reader to [**2**, Theorem 21]. Strict complementarity holds if $x_j^* + z_j^* \in \text{int}(\mathcal{K}_j)$ and implies that: (i) $x_j^* \in \text{int}(K_j) \implies z_j^* = 0$; (ii) $z_j^* \in \text{int}(K_j) \implies x_j^* = 0$; (iii) $x_j^* \in \text{bd}(K_j) \setminus \{0\} \iff z_j^* \in \text{bd}(K_j) \setminus \{0\}$; and (iv) not both x_j^* and z_j^* are zero.

2.3. Algorithm

The proposed algorithm solves the NLP formulation (2.3) using a variation of the SQP method. Since the functional formulation of the cone constraints (2.3c) might not be differentiable at all iterates or at an optimal solution, the cones are approximated by a polyhedral outer approximation using supporting hyperplanes.

The approximation is done so that the method implicitly identifies the constraints that are extremal-active at an optimal solution x^*, i.e., $\mathcal{E}(x^*) = \mathcal{E}(x^k)$ for large k. More precisely, we will show that close to a non-degenerate optimal solution, the steps generated by the proposed algorithm are identical to those computed by the QP subproblem (2.5) for the basic SQP algorithm for solving (2.4). Consequently, fast local quadratic convergence is achieved, as discussed in Chapter 2.2.2.

2.3.1. Supporting hyperplanes

In the following, consider a particular cone $\mathcal{K}_j$ and let $\mathcal{Y}_j$ be a finite subset of $\{y_j \in \mathbb{R}^{n_j} : \bar{y}_j \neq 0, y_{j0} \geq 0\}$. We define the cone

$$\mathcal{C}_j(\mathcal{Y}_j) = \left\{x_j \in \mathbb{R}^{n_j} : x_{j0} \geq 0 \text{ and } \nabla r_j(y_j)^T x_j \leq 0 \text{ for all } y_j \in \mathcal{Y}_j\right\} \tag{2.14}$$

generated by the points in $\mathcal{Y}_j$. For each $x_j \in \mathcal{K}_j$ we have $r_j(x_j) \leq 0$, and using

$$\nabla r_j(x_j) = \left(-1, \frac{\bar{x}_j^T}{\|\bar{x}_j\|}\right)^T, \tag{2.15}$$

we obtain for any $y_j \in \mathcal{Y}_j$ that

$$\nabla r_j(y_j)^T x_j = \frac{1}{\|\bar{y}_j\|}\bar{y}_j^T \bar{x}_j - x_{j0} \leq \frac{1}{\|\bar{y}_j\|}\|\bar{y}_j\|\|\bar{x}_j\| - x_{j0} = r_j(x_j) \leq 0.$$

Therefore $\mathcal{C}_j(\mathcal{Y}_j) \supseteq \mathcal{K}_j$. Also, for $y_j \in \mathcal{Y}_j$, consider $x_j = (1, \bar{y}_j^T/\|\bar{y}_j\|)^T$. Then

$$\nabla r_j(y_j)^T x_j = \frac{\bar{y}_j^T}{\|\bar{y}_j\|}\frac{\bar{y}_j}{\|\bar{y}_j\|} - 1 = 1 - 1 = 0,$$

and also $r_j(x_j) = \|\bar{x}_j\| - x_{j0} = \bar{y}_j/\|\bar{y}_j\| - 1 = 0$. Hence $x_j \in \mathcal{C}_j(\mathcal{Y}_j) \cap \mathcal{K}_j$. Therefore, for any $y_j \in \mathcal{Y}_j$, the inequality

$$\nabla r_j(y_j)^T x_j \leq 0 \tag{2.16}$$

defines a hyperplane that supports $\mathcal{K}_j$ at $(1, \bar{y}_j/\|\bar{y}_j\|)$. In summary, $\mathcal{C}_j(\mathcal{Y}_j)$ is a polyhedral outer approximation of $\mathcal{K}_j$, defined by supporting hyperplanes.

In addition, writing $\mathcal{Y}_j = \{y_{j,1}, \ldots, y_{j,m}\}$, we also define the cone

$$\mathcal{C}_j^\circ(\mathcal{Y}_j) := \left\{ -\sum_{l=1}^{m} \sigma_{j,l} \nabla r_j(y_{j,l}) + \eta_j e_{j0} : \sigma_j \in \mathbb{R}_+^m, \eta_j \geq 0 \right\}. \tag{2.17}$$

For all $x_j \in \mathcal{C}_j(\mathcal{Y}_j)$ and $z_j = -\sum_{l=1}^m \sigma_{j,l} \nabla r_j(y_{j,l}) + \eta_j e_{j0} \in \mathcal{C}_j^\circ(\mathcal{Y}_j)$, we have

$$x_j^T z_j = -\sum_{l=1}^{m} \sigma_{j,l} \nabla r_j(y_{j,l})^T x_j + \eta_j x_{j0} \geq 0$$

because $\nabla r_j(y_{j,l})^T x_j \leq 0$ and $x_{j0} \geq 0$ from the definition of $\mathcal{C}_j(\mathcal{Y}_j)$. Therefore $\mathcal{C}_j^\circ(\mathcal{Y}_j)$ is included in the dual of the cone $\mathcal{C}_j(\mathcal{Y}_j)$.

Now define $R = [-\nabla r_j(y_{j,1}), \ldots, -\nabla r_j(y_{j,m}), e_{j0}]$ and let $z_j \in \mathbb{R}^{n_j}$ be in the dual of $\mathcal{C}_j(\mathcal{Y}_j)$. Since this implies that $x_j^T z_j \geq 0$ for all $x \in \mathcal{C}_j(\mathcal{Y}_j) = \{\mathbb{R}^{n_j} : R^T x_j \geq 0\}$, Farkas' lemma yields that $z_j = R \cdot (\sigma^T, \eta)^T$ for some $\sigma_j \in \mathbb{R}_+^m$ and $\eta_j \geq 0$, i.e., $z_j \in \mathcal{C}_j^\circ(\mathcal{Y}_j)$.

Overall we proved that $\mathcal{C}_j^\circ(\mathcal{Y}_j)$ defined in (2.17) is the dual of $\mathcal{C}_j(\mathcal{Y}_j)$, and since $\mathcal{C}_j(\mathcal{Y}_j) \supseteq \mathcal{K}_j$, this implies $\mathcal{C}_j^\circ(\mathcal{Y}_j) \subseteq \mathcal{K}_j$.

Algorithm 2 Preliminary SQP Algorithm

Require: Initial iterate x^0 and sets $\mathcal{Y}_j^0$ for $j \in \mathcal{J}$.
1: **for** $k = 0, 1, 2 \cdots$ **do**
2: Choose H^k.
3: Solve subproblem (2.18) to get step d^k.
4: Set $x^{k+1} \leftarrow x^k + d^k$.
5: Set $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{pr,j}^+(\mathcal{Y}_j^k, x_j^k)$ for $j \in \mathcal{J}$.
6: **end for**

2.3.2. QP subproblem

In each iteration, at an iterate x^k, the proposed algorithm computes a step d^k as an optimal solution of the subproblem

$$\min_{d \in \mathbb{R}^n} \; c^T d + \tfrac{1}{2} d^T H^k d \tag{2.18a}$$

$$\text{s.t. } A(x^k + d) \leq b, \tag{2.18b}$$

$$r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j \leq 0, \qquad j \in \mathcal{D}(x^k), \tag{2.18c}$$

$$x_j^k + d_j \in \mathcal{C}_j(\mathcal{Y}_j^k), \qquad j \in \mathcal{J}. \tag{2.18d}$$

Here, H^k is a positive semi-definite matrix that captures the curvature of the nonlinear constraint (2.3c), and for each cone, $\mathcal{Y}_j^k$ is the set of hyperplane-generating points that have been accumulated up to this iteration. From (2.14), we see that (2.18d) can be replaced by linear constraints. Consequently, (2.18) is a QP and can be solved as such.

Algorithm 2 describes a preliminary version of the proposed SQP method based on this subproblem. Observe that the linearization (2.18c) can be rewritten as

$$0 \geq r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j = \|\bar{x}_j^k\| - x_{j0}^k - d_{j0} + \frac{(\bar{x}_j^k)^T \bar{d}_j}{\|\bar{x}_j^k\|}$$

$$= \frac{1}{\|\bar{x}_j^k\|}(\bar{x}_j^k)^T(\bar{x}_j^k + \bar{d}_j) - (x_{j0}^k + d_{j0}) = \nabla r_j(x_j^k)^T(x_j^k + d_j)$$

and is equivalent to the hyperplane constraint generated at x_j^k. Consequently, if $x_j^k \notin \mathcal{K}_j$, then $r_j(x_j^k) > 0$ and (2.18c) acts as a cutting plane that excludes x_j^k. Using the update rule

$$\mathcal{Y}_{pr,j}^{+}(\mathcal{Y}_j, x_j) = \begin{cases} \mathcal{Y}_j \cup \{x_j\} & \text{if } \bar{x}_j \neq 0 \text{ and } r_j(x_j) > 0, \\ \mathcal{Y}_j & \text{otherwise,} \end{cases} \tag{2.19}$$

in Step 5 makes sure that x_j^k is excluded in all future iterations.

In our algorithm, we initialize $\mathcal{Y}_j^0$ so that

$$\mathcal{Y}_j^0 \supseteq \hat{\mathcal{Y}}_j^0 := \{e_{ji} : i = 1, \ldots, n_j - 1\} \cup \{-e_{ji} : i = 1, \ldots, n_j - 1\}. \tag{2.20}$$

In this way, $x_j = 0$ is an extreme point of $\mathcal{C}_j(\mathcal{Y}_j^0)$, as it is for $\mathcal{K}_j$, and the challenging aspect of the cone is already captured in the first subproblem. By choosing the coordinate vectors e_{ji} we have $\nabla r_j(e_{ji})^T x_j = x_{ji} - x_{j0}$, and the hyperplane constraint (2.16) becomes a very sparse linear constraint.

When $H^k = 0$ in each iteration, this procedure becomes the standard cutting plane algorithm for the SOCP (2.1). It is well-known that the cutting plane algorithm is convergent in the sense that every limit point of the iterates is an optimal solution of the SOCP (2.1), but the convergence is typically slow. In the following chapters, we describe how Algorithm 2 is augmented to achieve fast local convergence. The full method is stated formally in Algorithm 3.

2.3.3. Identification of extremal-active cones

We now describe a strategy that enables our algorithm to identify those cones that are extreme-active at a non-degenerate solution x^* within a finite number of iterations, i.e.,

$\mathcal{E}(x^k) = \mathcal{E}(x^*)$ for all large k. This will make it possible to apply a second-order method and achieve quadratic local convergence.

Consider the optimality conditions for the QP subproblem (2.18):

$$c + H^k d^k + A^T \hat{\lambda}^k + \sum_{j \in \mathcal{D}(x^k)} \hat{\mu}_j^k \nabla_x r_j(x^k) - \hat{\nu}^k = 0, \tag{2.21a}$$

$$A(x^k + d^k) - b \leq 0 \perp \hat{\lambda}^k \geq 0, \tag{2.21b}$$

$$r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j^k \leq 0 \perp \hat{\mu}_j \geq 0, \qquad j \in \mathcal{D}(x^k), \tag{2.21c}$$

$$\mathcal{C}_j(\mathcal{Y}_j^k) \ni x_j^k + d_j^k \perp \hat{\nu}_j^k \in \mathcal{C}_j^\circ(\mathcal{Y}_j^k), \quad j \in \mathcal{J}. \tag{2.21d}$$

Here, $\hat{\lambda}^k$, $\hat{\mu}_j^k$, and $\hat{\nu}_j^k$ are the multipliers corresponding to the constraints in (2.18); for completeness, we define $\hat{\mu}_j^k = 0$ for $j \in \mathcal{J} \setminus \mathcal{D}(x^k)$. In (2.21a), $\nabla_x r_j(x^k)$ is the vector in $\mathbb{R}^n$ that contains $\nabla r_j(x_j^k)$ in the elements corresponding to x_j and is zero otherwise. Similarly, $\hat{\nu}^k \in \mathbb{R}^n$ is equal to $\hat{\nu}_j^k$ in the elements corresponding to x_j for all $j \in \mathcal{J}$ and zero otherwise.

Let us define

$$\hat{\mathcal{Y}}_j^k := \begin{cases} \mathcal{Y}_j^k \cup \{x_j^k\}, & \text{if } j \in \mathcal{D}(x^k), \\ \mathcal{Y}_j^k, & \text{if } j \in \mathcal{J} \setminus \mathcal{D}(x^k). \end{cases} \tag{2.22}$$

It is easy to verify that, for $j \in \mathcal{D}(x^k)$, $\nabla r_j(x_j^k) x_j^k = r_j(x_j^k)$ and hence $r_j(x_j^k)^T (x_j^k + d^k) \leq 0$ from (2.21c). As a consequence we obtain $x_j^k + d_j^k \in \mathcal{C}_j(\hat{\mathcal{Y}}_j^k)$ for all $j \in \mathcal{J}$. Furthermore, $\hat{\nu}_j^k \in \mathcal{C}_j^\circ(\mathcal{Y}_j^k)$ implies that

$$\hat{\nu}_j^k = -\sum_{l=1}^{m} \sigma_{j,l}^k \nabla r_j(y_{j,l}^k) + \eta_j^k e_{j0}$$

for suitable values of $\sigma_{j,l}^k \geq 0$ and $\eta_j^k \geq 0$. Then $\hat{z}_j^k := -\hat{\mu}_j^k \nabla r_j(x^k) + \hat{\nu}_j^k \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k)$ and

$$\hat{z}^k = c + H^k d^k + A^T \hat{\lambda}^k \tag{2.23}$$

from (2.21a). In conclusion, if $(d, \hat{\lambda}^k, \hat{\mu}^k, \hat{\nu}^k)$ is a primal-dual solution of the QP subproblem (2.18), then $(d, \hat{\lambda}^k, \hat{z}^k)$ satisfies the conditions

$$c + H^k d^k + A^T \hat{\lambda}^k - \hat{z}^k = 0, \tag{2.24a}$$

$$A(x^k + d^k) - b \leq 0 \perp \hat{\lambda}^k \geq 0, \tag{2.24b}$$

$$\mathcal{C}_j(\hat{\mathcal{Y}}_j^k) \ni x_j^k + d_j^k \perp \hat{z}_j^k \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k), \quad j \in \mathcal{J}, \tag{2.24c}$$

which more closely resembles the SOCP optimality conditions (2.13). Our algorithm maintains primal-dual iterates $(x^{k+1}, \hat{\lambda}^k, \hat{z}^k)$ that are updated based on (2.24).

Suppose that strict-complementarity holds at a primal-dual solution (x^*, λ^*, z^*) of the SOCP (2.1) and that $(x^{k+1}, \hat{\lambda}^k, \hat{z}^k) \to (x^*, \lambda^*, z^*)$. If $j \notin \mathcal{E}(x^*)$ then $x_j^* \in \mathcal{K}_j$ implies $x_{j0}^* > 0$. As x_j^k converges to x_j^*, we have $x_{j0}^k > 0$ and therefore $j \notin \mathcal{E}(x^k)$ for sufficiently large k. This yields $\mathcal{E}(x^k) \subseteq \mathcal{E}(x^*)$. We now derive a modification of Algorithm 2 that ensures that $\mathcal{E}(x^*) \subseteq \mathcal{E}(x^k)$ for all sufficiently large k under Assumption 2.2.2.

Consider any $j \in \mathcal{E}(x^*)$. We would like to have

$$\hat{z}_j^k \in \operatorname{int}(\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k)) \tag{2.25}$$

for all large k, since then complementarity in (2.24c) implies that $x_j^{k+1} = x_j^k + d_j^k = 0$ and hence $j \in \mathcal{E}(x^{k+1})$ for all large k. We will later show that Assumption 2.2.2 implies that $\hat{z}_j^k \to z_j^*$ and that there exists a neighborhood $N_\epsilon(z_j^*) = \{z_j \in \mathbb{R}^{n_j} : \|z_j - z_j^*\| \leq \epsilon\}$ of z_j^* so that $z_j \in \operatorname{int}(\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^0 \cup \{-y_j\}))$ if $z_j, y_j \in N_\epsilon(z_j^*)$; see Remark 2.4.2. This suggests that some

vector close to $-z_j^*$ should eventually be included in $\hat{\mathcal{Y}}_j^k$ because then (2.25) holds when $\hat{z}_j^k$ is close enough to z_j^*. For this purpose, the algorithm computes

$$\check{z}^k = c + A^T \hat{\lambda}^k,$$

which also converges to z_j^* (see (2.13a)), and sets $\mathcal{Y}_j^{k+1}$ to $\mathcal{Y}_{du,j}^+(\mathcal{Y}_j^k, x_j^k, \check{z}_j^k)$, where

$$\mathcal{Y}_{du,j}^+(\mathcal{Y}_j, x_j, z_j) = \begin{cases} \mathcal{Y}_j \cup \{-z_j\} & \text{if } x_j \neq 0, \bar{z}_j \neq 0 \text{ and } r_j(z_j) < 0, \\ \mathcal{Y}_j & \text{otherwise.} \end{cases} \tag{2.26}$$

The update is skipped when $x_j^k = 0$ (because then j is already in $\mathcal{E}(x^k)$ and no additional hyperplane is needed), and when $\bar{\check{z}}_j^k = 0$ or $r_j(\check{z}_j^k) \geq 0$, which might indicate that $z_j^* \notin \operatorname{int}(\mathcal{K}_j)$ and $j \notin \mathcal{E}(x^*)$.

2.3.4. Fast NLP-SQP steps

Now that we have a mechanism in place that makes sure that the extremal-active cones are identified in a finite number of iterations, we present a strategy that emulates the basic SQP Algorithm 1 and automatically takes quadratically convergent SQP steps, i.e., solutions of the SQP subproblem (2.5), close to x^*. For the discussion in this chapter, we again assume that x^* is a unique solution at which Assumption 2.2.2 holds.

Suppose that $\mathcal{E}(x^k) = \mathcal{E}(x^*)$ for large k due to the strategy discussed in Chapter 2.3.3. This means that the outer approximation (2.18d) of $\mathcal{K}_j$ for $j \in \mathcal{E}(x^*)$ is sufficient to fix x_j^k to zero and is therefore equivalent to the constraint (2.5d) in the basic SQP subproblem. However, (2.18) includes the outer approximations for all cones, including those for $j \notin \mathcal{E}(x^*)$, which are not present in (2.5). Consequently, the desired SQP step from (2.5) might not be feasible for (2.18).

As a remedy, at the beginning of an iteration, the algorithm first computes an NLP-SQP step as an optimal solution $d^{S,k}$ of a relaxation of (2.18),

$$\min_{d\in\mathbb{R}^n} \; c^T d + \tfrac{1}{2} d^T H^k d \tag{2.27a}$$

$$\text{s.t. } A(x^k + d) \le b \tag{2.27b}$$

$$r_j(x_j^k) + \nabla r_j(x_j^k)^T d_j \le 0, \qquad j \in \mathcal{D}(x^k) \tag{2.27c}$$

$$x_{j0}^k + d_{j0} \ge 0, \qquad j \in \mathcal{D}(x^k) \setminus \hat{\mathcal{E}}^k \tag{2.27d}$$

$$x_j^k + d_j \in \mathcal{C}_j(\mathcal{Y}_j^k) \qquad j \in \hat{\mathcal{E}}^k, \tag{2.27e}$$

where $\hat{\mathcal{E}}^k = \mathcal{E}(x^k)$. In this way, the outer approximations are imposed only for the currently extremal-active cones, while for all other cones only the linearization (2.27c) is considered, just like in (2.5), with the additional restriction (2.27d) that ensure $x_{j0}^{k+1} \ge 0$. Let $\hat{\lambda}^k$, $\hat{\mu}_j^k$, $\hat{\eta}_j^k$, and $\hat{\nu}_j^k$ be the optimal corresponding to the constraints in (2.27) (set to zero for non-existing constraints) and define $\hat{z}^k$ as in (2.23). Then the optimality conditions (2.24) hold again, this time with $d^k = d^{S,k}$, but instead of (2.22) we have

$$\hat{\mathcal{Y}}_j^k := \begin{cases} \{x_j^k\} & \text{if } j \in \mathcal{D}(x^k) \setminus \hat{\mathcal{E}}^k, \\ \mathcal{Y}_j^k \cup \{x_j^k\} & \text{if } j \in \hat{\mathcal{E}}^k \cap \mathcal{D}(x^k), \\ \mathcal{Y}_j^k & \text{if } j \in \hat{\mathcal{E}}^k \setminus \mathcal{D}(x^k). \end{cases} \tag{2.28}$$

When x^k is not close to x^* and $\mathcal{E}(x^*) \neq \mathcal{E}(x^k)$, QP (2.27) might result in poor steps that go far outside of $\mathcal{K}_j$ for some $j \in \mathcal{D}(x^k) \setminus \hat{\mathcal{E}}^k$ and undermine convergence. Therefore, we iteratively add more cones to $\hat{\mathcal{E}}^k$ until

$$x_{j0}^k + d_{j0}^{S,k} > 0 \text{ only for } j \in \mathcal{J} \setminus \hat{\mathcal{E}}^k, \tag{2.29}$$

i.e., when a cone is approximated only by its linearization (2.27c), the step does not appear to target its extreme point. This property is necessary to show that $\mathcal{E}(x^k) = \mathcal{E}(x^*)$ for all large k also for the case that new iterates are computed from (2.27) instead of (2.18). Note that in the extreme case $\hat{\mathcal{E}}^k = \mathcal{J}$ and (2.27) is identical to (2.18). This loop can be found in Steps 6–9 in Algorithm 3.

Since there is no guarantee that (2.27) yields iterates that converge to x^*, the algorithm discards the NLP-SQP step in certain situations and falls back to the original method to recompute a new step from (2.18), as in the original method. In Chapter 2.3.6 we describe how we use the exact penalty function (2.8) to determine when this is necessary.

2.3.5. Hessian matrix

Motivated by (2.6), we compute the Hessian matrix H^k in (2.18) and (2.27) from

$$H^k = \sum_{j \in \mathcal{D}(x^k)} \mu_j^k \nabla_{xx}^2 r_j(x^k), \tag{2.30}$$

where $\mu_j^k \geq 0$ are multiplier estimates for the nonlinear constraint (2.3c). Because $\nabla^2 r_j(x_j^k)$ is positive semi-definite and $\mu_j^k \geq 0$, also H^k is positive semi-definite.

In the final phase, when we intend to emulate the basic SQP Algorithm 1. Therefore, we set $\mu_j^{k+1} = \hat{\mu}_j^k$ for $j \in \mathcal{D}(x^k)$, where $\hat{\mu}_j^k$ are the optimal multipliers for (2.27c), when the fast NLP-SQP step was accepted. But we also need to define a value for μ_j^{k+1} when the step is computed from (2.18) where, in addition to the linearization of r_j, hyperplanes (2.18d) are used to approximate all cones. By comparing the optimality conditions of the QPs (2.18) and (2.5) we now derive an update for μ_j^{k+1}.

Suppose that $j \in \mathcal{D}(x^{k+1}) \cap \mathcal{D}(x^k)$. Then (2.21a) yields

$$c_j + H_{jj}^k d_j^k + A_j^T \hat{\lambda}^k + \hat{\mu}_j^k \nabla r_j(x_j^k) - \hat{\nu}_j^k = 0, \tag{2.31}$$

where $H_{jj}^k = \mu_j^k \nabla^2 r_j(x_j^k)$ because of (2.30). Here, the dual information for the nonlinear constraint is split into $\hat{\mu}_j^k$ and $\hat{\nu}_j^k$ and needs to be condensed into a single number, μ_j^{k+1}, so that we can compute H^k from (2.30) in the next iteration.

Recall that, in the basic SQP Algorithm 1, the new multipliers μ_j^{k+1} are set to the optimal multipliers of the QP (2.5), which satisfy

$$c_j + H_{jj}^k d_j^k + A_j^T \hat{\lambda}^k + \mu_j^{k+1} \nabla r_j(x_j^k) = 0. \tag{2.32}$$

A comparison with (2.31) suggests to choose μ_j^{k+1} so that $\mu_j^{k+1} \nabla r_j(x_j^k) \approx \hat{\mu}_j^k \nabla r_j(x_j^k) - \hat{\nu}_j^k$. Multiplying both sides with $\nabla r_j(x_j^k)^T$ and solving for μ^{k+1} yields

$$\mu_j^{k+1} = \hat{\mu}_j^k - \frac{\nabla r_j(x_j^k)^T \hat{\nu}_j^k}{\|\nabla r_j(x_j^k)\|^2}.$$

Note that $\mu_j^{k+1} = \hat{\mu}_j^k$ if the outer approximation constraint (2.18d) is not active and therefore $\hat{\nu}_j^k = 0$ for j. In this case, we recover the basic SQP update, as desired.

Now suppose that $j \in \mathcal{D}(x^{k+1}) \setminus \mathcal{D}(x^k)$. Again comparing (2.31) with (2.32) suggests a choice so that $\mu_j^{k+1} \nabla r_j(x_j^{k+1}) \approx -\hat{\nu}_j^k$, where we substituted $\nabla r_j(x_j^k)$ by $\nabla r_j(x_j^{k+1})$ because the former is not defined for $j \notin \mathcal{D}(x^k)$. In this case, multiplying both sides with $\nabla r_j(x_j^{k+1})^T$ and solving for μ^{k+1} yields

$$\mu_j^{k+1} = -\frac{\nabla r_j(x_j^{k+1})^T \hat{\nu}_j^k}{\|\nabla r_j(x_j^{k+1})\|^2}.$$

In summary, in each iteration in which (2.18) determines the new iterate, we update

$$\mu_j^{k+1} = \begin{cases} \hat{\mu}_j^k - \frac{\nabla r_j(x_j^k)^T \hat{\nu}_j^k}{\|\nabla r_j(x_j^k)\|^2} & j \in \mathcal{D}(x^{k+1}) \cap \mathcal{D}(x^k) \\ -\frac{\nabla r_j(x_j^{k+1})^T \hat{\nu}_j^k}{\|\nabla r_j(x_j^{k+1})\|^2} & j \in \mathcal{D}(x^{k+1}) \setminus \mathcal{D}(x^k) \\ 0 & \text{otherwise.} \end{cases} \tag{2.33}$$

The choice above leads to quadratic convergence for non-degenerate instances, but it is common for the global convergence analysis of SQP methods to permit any positive semi-definite Hessian matrix H^k, as long as it is bounded. Since we were not able to exclude the case that μ_j^k or $1/x_{j0}^k$ are unbounded for some cone $j \in \mathcal{J}$, in which case H^k defined in (2.30) is unbounded, we fix a large threshold $c_H > 0$ and rescale the Hessian matrix according to

$$H^k \leftarrow H^k \cdot \min\{1, c_H / \|H^k\|\} \tag{2.34}$$

so that $\|H^k\| \leq c_H$ in every iteration. In this way, global convergence is guaranteed, but the fast local convergence rate might be impaired if c_H is chosen too small so that H^k defined in (2.30) must be rescaled. Therefore, in practice, we set c_H to a very large number and (2.34) is never actually triggered in our numerical experiments.

2.3.6. Penalty function

The steps computed from (2.18) and (2.27) do not necessarily yield a convergent algorithm and a safeguard is required to force the iterates into a neighborhood of an optimal solution. Here, we utilized the exact penalty function (2.8) and accept a new iterate only if the sufficient decrease condition (2.11) holds.

As discussed in Chapter 2.3.4, at the beginning of an iteration, the algorithm first computes an NLP-SQP step $d^{S,k}$ from (2.27). The penalty function can now help us to decide whether this step makes sufficient progress towards an optimal solution, and we only accept the trial point $\hat{x}^{k+1} = x^k + d^{S,k}$ as a new iterate if (2.11) holds with $d = d^{S,k}$.

If the penalty function does not accept $d^{S,k}$, there is still a chance that $d^{S,k}$ is making rapid progress towards the solution, but, as discussed in Chapter 2.2.2, the Maratos effect is preventing the acceptance of $d^{S,k}$. As a remedy, we compute, analogously to (2.12), a second-order correction step s^k for (2.27) as a solution of

$$
\begin{aligned}
&\min_{s\in\mathbb{R}^n} \; c^T(d^{S,k}+s) + \tfrac{1}{2}(d^{S,k}+s)^T H^k (d^{S,k}+s) \\
&\text{s.t. } A(x^k + d^{S,k} + s) \le b, \\
&\qquad r_j(x_j^k + d_j^{S,k}) + \nabla r_j(x_j^k + d_j^{S,k})^T s_j \le 0, \qquad j \in \mathcal{D}(x^k), \\
&\qquad x_{j0}^k + d_{j0}^{S,k} + s_{j0} \ge 0, \qquad\qquad j \in \mathcal{D}(x^k) \setminus \hat{\mathcal{E}}^k, \\
&\qquad x_j^k + d_j^{S,k} + s_j \in \mathcal{C}_j(\mathcal{Y}_j^k), \qquad\qquad j \in \hat{\mathcal{E}}^k,
\end{aligned}
\tag{2.35}
$$

and accept the trial point $\hat{x}^{k+1} = x^k + d^{S,k} + s^k$ if it satisfies (2.11) with $d = d^{S,k}$. Let again $\hat{\lambda}^k$, $\hat{\mu}_j^k$, $\hat{\eta}_j^k$, and $\hat{\nu}_j^k$ denote the optimal multipliers in (2.35) and define $\hat{z}^k$ as in (2.23). The optimality conditions (2.24) still hold, this time with $d^k = d^{S,k} + s^k$ and

$$
\hat{\mathcal{Y}}_j^k := \begin{cases} \{x_j^k + d_j^{S,k}\}, & \text{if } j \in \mathcal{D}(x^k) \setminus \hat{\mathcal{E}}^k, \\ \mathcal{Y}_j^k \cup \{x_j^k + d_j^{S,k}\}, & \text{if } j \in \mathcal{D}(x^k) \cap \hat{\mathcal{E}}^k, \\ \mathcal{Y}_j^k, & \text{if } j \in \hat{\mathcal{E}}^k. \end{cases}
\tag{2.36}
$$

If neither $d^{S,k}$ nor $d^{S,k} + s^k$ has been accepted, we give up on fast NLP-SQP steps and instead revert to QP (2.18) which safely approximates every cone with an outer approximation. However, the trial point $\hat{x}^{k+1} = x^k + d^k$ with the step d^k obtained from (2.18) does not necessarily satisfy (2.11). In that case, the algorithm adds $x^k + d^k$ to $\mathcal{Y}_j^k$ to cut off $x^k + d^k$ and resolves (2.18) to get a new trial step d^k. In an inner loop (Steps 21–31), this procedure is repeated until, eventually, a trial step is obtained that satisfies (2.11). We will show that (2.11) holds after a finite number of iterations of the inner loop.

It remains to discuss the update of the penalty parameter estimate ρ^k. One can show (see Lemma 2.4.1) that an optimal solution of x^* of the SOCP with multipliers z^* is a minimizer of $\phi(\,\cdot\,, \rho)$ over the set X defined in (2.9) if $\rho > \|z^*_{\mathcal{J},0}\|_\infty$, where $z^*_{\mathcal{J},0} = (z^*_{1,0}, \ldots, z^*_{p,0})^T$. Since z^* is not known *a priori*, the algorithm uses the update rule $\rho^k = \rho_{\text{new}}(\rho^{k-1}, z^k)$ where

$$\rho_{\text{new}}(\rho_{\text{old}}, z) := \begin{cases} \rho_{\text{old}} & \text{if } \rho_{\text{old}} > \|z_{\mathcal{J},0}\|_\infty \\ c_{\text{inc}} \cdot \|z_{\mathcal{J},0}\|_\infty & \text{otherwise,} \end{cases} \tag{2.37}$$

with $c_{\text{inc}} > 1$. We will prove in Lemma 2.4.5 that the sequence $\{z^k\}_{k=1}^\infty$ is bounded under Slater's constraint qualification. Therefore, this rule will eventually settle at a final penalty parameter ρ^∞ that is not changed after a finite number of iterations.

During an iteration of the algorithm, several trial steps may be considered and a preliminary parameter value is computed from (2.37) for each one. At the end of the iteration, the parameter value corresponding to the accepted trial step is stored. Note that the acceptance test for the second-order correction step from (2.35) needs to be done with the penalty parameter computed for the regular NLP-SQP step from (2.27).

Algorithm 3 SQP Algorithm for SOCP.

Require: Initial iterate $x^0 \in X$ with $x_{j,0} \geq 0$, multipliers $\mu_j^0 \in \mathbb{R}_+$, penalty parameter $\rho^{-1} > 0$; constants $c_{\text{dec}} \in (0,1)$, $c_{\text{inc}} > 1$, and $c_H > 0$.

Initialize $\mathcal{Y}_j^0$ so that (2.20) is satisfied.
for $k = 0, 1, 2, \ldots$ **do**
  Compute H^k using (2.30). Rescale according to (2.34) if $\|H^k\| > c_H$.
  Set $\hat{\mathcal{E}}^k \leftarrow \mathcal{E}(x^k)$.
  Compute $d^{S,k}$, $\hat{\lambda}^k$, $\hat{\mu}^k$, $\hat{z}^k$ from (2.27) and (2.23) and set $\hat{x}^{k+1} = x^k + d^{S,k}$.
  while $\{j \in \mathcal{J} : x_{j0}^k + d_{j0}^{S,k} = 0\} \not\subseteq \hat{\mathcal{E}}^k$ **do**
    Set $\hat{\mathcal{E}}^k \leftarrow \hat{\mathcal{E}}^k \cup \{j \in \mathcal{J} : x_{j0}^k + d_{j0}^{S,k} = 0\}$.
    Recompute $d^{S,k}$, $\hat{\lambda}^k$, $\hat{\mu}^k$, $\hat{z}^k$ from (2.27) and (2.23) and set $\hat{x}^{k+1} = x^k + d^{S,k}$.
  end while
  Compute candidate penalty parameter $\rho^k = \rho_{\text{new}}(\rho^{k-1}, \hat{z}^k)$, see (2.37).
  if (2.11) holds for $d = d^{S,k}$ **then**
    Set $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{pr,j}^+(\mathcal{Y}_j^k, x_j^k)$ using (2.19) and set $d^k = d^{S,k}$.
    Set $\mu^{k+1} = \hat{\mu}^k$ and go to Step 33.
  end if
  Compute s^k, $\hat{\lambda}^k$, $\hat{\mu}^k$, $\hat{z}^k$ from (2.35) and (2.23) and set $\hat{x}^{k+1} = x^k + d^{S,k} + s^k$.
  if (2.11) holds for $d = d^{S,k}$ and $\{j \in \mathcal{J} : x_{j0}^k + d_{j0}^{S,k} + s^k = 0\} \subseteq \hat{\mathcal{E}}^k$ **then**
    Set $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{pr,j}^+(\mathcal{Y}_j^k, x_j^k)$ and $d^k = d^{S,k}$.
    Set $\mu^{k+1} = \hat{\mu}^k$ and go to Step 33.
  end if
  Set $\mathcal{Y}_j^{k,0} \leftarrow \mathcal{Y}_j^k$.
  for $l = 0, 1, 2, \ldots$ **do**
    Compute $d^{k,l}$, $\hat{\lambda}^k$, $\hat{\mu}^k$, $\hat{z}^k$ from (2.18) and (2.23) and set $\hat{x}^{k+1} = x^k + d^{k,l}$.
    Compute candidate penalty parameter $\rho^k = \rho_{\text{new}}(\rho^{k-1}, \hat{z}^k)$.
    if (2.11) holds for $d = d^{k,l}$ **then**
      Set $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{pr,j}^+(\mathcal{Y}_j^{k,l}, x_j^k)$ and $d^k = d^{k,l}$.
      Go to Step 32.
    end if
    Set $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{pr,j}^+(\mathcal{Y}_j^{k,l}, \hat{x}_j^{k+1})$, see (2.19).
    Compute $\check{z}^k = c + A^T\hat{\lambda}^k$.
    Update $\mathcal{Y}_j^{k,l+1} \leftarrow \mathcal{Y}_{du,j}^+(\mathcal{Y}_j^{k,l+1}, \hat{x}_j^{k+1}, \check{z}_j^k)$, see (2.26).
  end for
  Compute μ^{k+1} from (2.33).
  Compute $\check{z}^k = c + A^T\hat{\lambda}^k$ and update $\mathcal{Y}_j^{k+1} \leftarrow \mathcal{Y}_{du,j}^+(\mathcal{Y}_j^{k+1}, x_j^k, \check{z}_j^k)$.
  Set $x^{k+1} \leftarrow \hat{x}^{k+1}$.
  If $(x^{k+1}, \hat{\lambda}^k, \hat{z}^k)$ satisfy (2.13), **stop**.
end for

2.3.7. Complete algorithm

The complete method is stated in Algorithm 3. To keep the notation concise, we omit "for all $j \in \mathcal{J}$" whenever the index j is used. We assume that all QPs in the algorithm are solved exactly.

Each iteration begins with the computation of the fast NLP-SQP step where an inner loop repeatedly adds cones to $\hat{\mathcal{E}}^k$ until (2.29) holds. If the step achieves a sufficient decrease in the penalty function, the trial point is accepted. Otherwise, the second-order correction for the NLP-SQP step is computed and accepted if it yields a sufficient decrease for the NLP-SQP step. Note that the second-order correction step is discarded if it does not satisfy (2.29) since otherwise finite identification of $\mathcal{E}(x^*)$ cannot be guaranteed. If none of the NLP-SQP steps was acceptable, the algorithm proceeds with an inner loop in which hyperplanes cutting off the current trial point are repeatedly added until the penalty function is sufficiently decreased. No matter which step is taken, both x_j^k and $\check{z}_j^k$ are added to $\mathcal{Y}_j^k$ according to the update rules (2.19) and (2.26) and the multiplier μ^k for the nonlinear constraints is updated.

In most cases, a new QP is obtained by adding only a few constraints to the most recently solved QP, and a hot-started QP solver will typically compute the new solution quickly. For example, in each inner iteration in Steps 6–9, hyperplanes for the polyhedral outer approximation for cones augmenting $\hat{\mathcal{E}}^k$ are added to QP (2.27). Similarly, each inner iteration in Steps 21–31 adds one cutting plane for a violated cone constraint. In Steps 5 and 15, some constraints are removed compared to the most recently solved QP, but also this structure could be utilized.

The algorithm might terminate because one of QPs solved for the step computation is infeasible. Since the feasible regions of the QP are outer approximations of the SOCP (2.1), this proves that the SOCP instance is infeasible; see also Remark 2.4.1.

2.4. Convergence analysis

2.4.1. Global convergence

In this chapter, we prove that, under a standard regularity assumption, all limit points of the sequence of iterates are optimal solutions of the SOCP, if the algorithm does not terminate with an optimal solution in Step 35. We also explore what happens when the SOCP is infeasible.

We make the following assumption throughout this chapter.

Assumption 2.4.1. *The set X defined in* (2.9) *is bounded.*

Since $x^0 \in X$ by the initialization of Algorithm 3 and any step satisfies (2.21b), we have $x^k \in X$ for all k. Similarly, (2.24c) and (2.14) imply that

$$x^k_{j0} \geq 0 \text{ for all } k \geq 0 \text{ and } j \in \mathcal{J}. \tag{2.38}$$

We start the analysis with some technical results that quantify the decrease in the penalty function model.

Lemma 2.4.1. *Consider an iteration k and let d^k be computed in Step 5 or Step 22 in Algorithm 3. Further let $\rho^k > \rho^k_{\min}$, where $\rho^k_{\min} = \|\hat{z}^k_{\mathcal{J},0}\|_\infty$ with $\hat{z}^k$ defined in* (2.23)*. Then the following statements are true.*

(1) *We have*

$$m^k(x^k + d^k; \rho^k) - m^k(x^k; \rho^k) \leq -(d^k)^T H^k d^k - (\rho^k - \rho^k_{\min}) \sum_{j \in \mathcal{J}} [r_j(x^k_j)]^+ \leq 0.$$

(2) *If x^k is not an optimal solution of the SOCP, then*

$$m^k(x^k + d^k; \rho^k) - m^k(x^k; \rho^k) < 0. \tag{2.39}$$

Proof. Proof of (1): Consider any $j \in \mathcal{D}(x^k)$. Because d^k is a solution of (2.18) or (2.27), there exist $\hat{\lambda}^k$ and $\hat{z}^k$ so that the optimality conditions (2.24) hold. Let $j \in \mathcal{J}$. Since $\hat{z}_j^k \in \mathcal{C}^\circ(\hat{\mathcal{Y}}_j^k)$, the definition (2.17) implies that

$$\hat{z}_j^k = -\sum_{l=1}^{m_j^k} \hat{\sigma}_{l,j}^k \nabla r_j(y_{j,l}^k) + \hat{\eta}_j^k e_{j0},$$

where $\hat{\mathcal{Y}}_j^k = \left\{ y_{j,1}^k, \ldots, y_{j,m_j^k}^k \right\}$ and $\hat{\sigma}_{l,j}^k, \hat{\eta}_j^k \in \mathbb{R}_+$.

Using (2.15) we have $\hat{z}_{j0}^k = \sum_{l=1}^{m_j^k} \hat{\sigma}_{l,j}^k + \hat{\eta}_j^k \geq \sum_{l=1}^{m_j^k} \hat{\sigma}_{l,j}^k$. Together with $(x_j^k + d_j^k)^T \hat{z}_j^k = 0$ from (2.24c) and $(\bar{x}_j^k)^T \bar{y}_{l,j}^k \leq \|\bar{x}_j^k\| \cdot \|\bar{y}_{l,j}^k\|$ this overall yields

$$\begin{aligned} -(d_j^k)^T \hat{z}_j^k = (x_j^k)^T \hat{z}_j^k &= x_{j0}^k \hat{z}_{j0}^k - \sum_{l=1}^{m_j^k} \hat{\sigma}_{l,j}^k (\bar{x}_j^k)^T \frac{\bar{y}_{l,j}^k}{\|\bar{y}_{l,j}^k\|} \geq x_{j0}^k \hat{z}_{j0}^k - \sum_{l=1}^{m_j^k} \hat{\sigma}_{l,j}^k \|\bar{x}_j^k\| \\ &\geq x_{j0}^k \hat{z}_{j0}^k - \hat{z}_{j0}^k \|\bar{x}_j^k\| = -z_{j0}^k r_j(x_j^k) \geq -z_{j0}^k [r_j(x_j^k)]^+. \end{aligned}$$

Further, we have from (2.24b) that $0 = (Ax^k + Ad^k - b)^T \hat{\lambda}^k$ and therefore $(d^k)^T A^T \hat{\lambda}^k = -(Ax^k - b)^T \hat{\lambda}^k \geq 0$ since $\hat{\lambda}^k \geq 0$ and $x^k \in X$.

Using these inequalities and (2.24a), the choice of $\rho_{\min}^k$ yields

$$\begin{aligned} 0 &= (d^k)^T \left(c + H^k d^k + A^T \hat{\lambda}^k - \hat{z}^k \right) \\ &\geq c^T d^k + (d^k)^T H^k d^k - \sum_{j \in \mathcal{J}} \hat{z}_{j0}^k [r_j(x_j^k)]^+ \\ &\geq c^T d^k + (d^k)^T H^k d^k - \rho_{\min}^k \sum_{j \in \mathcal{J}} [r_j(x_j^k)]^+. \end{aligned}$$

Finally, combining this with (2.10) and (2.18c) or (2.27c), respectively, we obtain

$$\begin{aligned} m^k(x^k + d^k; \rho^k) - m^k(x^k; \rho^k) &= c^T d^k - \rho^k \sum_{j \in \mathcal{D}(x^k)} [r_j(x_j^k)]^+ \\ &= c^T d^k - \rho^k \sum_{j \in \mathcal{J}} [r_j(x_j^k)]^+ \\ &\leq -(d^k)^T H^k d^k - (\rho^k - \rho^k_{\min}) \sum_{j \in \mathcal{J}} [r_j(x_j^k)]^+. \end{aligned}$$

For the second equality, we used that $r_j(x_j^k) = 0 - x_{j0}^k \leq 0$ for $j \notin \mathcal{D}(x^k)$ by (2.38) and the definition of $\mathcal{D}(x^k)$. Since H^k is positive semi-definite, $\rho^k > \rho^k_{\min}$, and $[r_j(x_j^k)]^+ \geq 0$, the right-hand side must be non-positive.

Proof of (2): Suppose $x^k \in X$ is not an optimal solution for the SOCP. If x^k is not feasible for the SOCP, x^k must violate a conic constraint and we have $[r_j(x_j^k)]^+ > 0$ for some $j \in \mathcal{J}$. Since H^k is positive semidefinite and $\rho^k - \rho^k_{\min} > 0$, part (1) yields (2.39).

It remains to consider the case when x^k is feasible for the SOCP, i.e., $[r_j(x_j^k)]^+ = 0$ for all j. To derive a contradiction, suppose that (2.39) does not hold. Then part (1) yields

$$\begin{aligned} 0 &= m^k(x^k + d^k; \rho^k) - m^k(x^k; \rho^k) \\ &= -(d^k)^T H^k d^k - (\rho^k - \rho^k_{\min}) \sum_{j \in \mathcal{J}} [r_j(x_j^k)]^+ = -(d^k)^T H^k d^k \leq 0. \end{aligned}$$

Because H^k is positive semi-definite, this implies $H^k d^k = 0$. Further, since also

$$0 = m^k(x^k + d^k; \rho^k) - m^k(x^k; \rho^k) \overset{(2.10)}{=} c^T d^k - \rho^k \sum_{j \in \mathcal{D}(x^k)} [r_j(x_j^k)]^+ = c^T d^k,$$

the optimal objective value of (2.18) or (2.27), respectively, is zero. At the same time, choosing $d^k = 0$ is also feasible for (2.18) or (2.27) and yields the same objective value.

Therefore, also $d^k = 0$ is an optimal solution of (2.18) or (2.27) and the optimality conditions (2.24) hold for some multipliers. Because $\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_k^k) \subseteq \mathcal{K}_j$, the same multipliers and $d^k = 0$ show that the optimality conditions of the SOCP (2.13) also hold. So, x^k is an optimal solution for the SOCP, contradicting the assumption. □

The following lemma shows that the algorithm is well-defined and will not stay in an infinite loop in Steps 21–31.

Lemma 2.4.2. *Consider an iteration k and let d^k be computed in Step 5 or Step 22 in Algorithm 3. Suppose that x^k is not an optimal solution of the SOCP. Then*

$$\varphi(x^k + d^{k,l}; \rho^k) - \varphi(x^k; \rho^k) \leq c_{dec}\Big(m^k(x^k + d^{k,l}; \rho^k) - m^k(x^k; \rho^k)\Big) \tag{2.40}$$

after a finite number of iterations in the inner loop in Steps 21–31.

Proof. Suppose the claim is not true and let $\{d^{k,l}\}_{l=0}^\infty$ be the infinite sequence of trial steps generated in the loop in Steps 21–31 for which the stopping condition in Step 24 is never satisfied, and let $d^{k,\infty}$ be a limit point of $\{d^{k,l}\}_{l=0}^\infty$. We will first show that

$$[r_j(x_j^k + d_j^{k,\infty})]^+ = 0 \text{ for all } j \in \mathcal{J}. \tag{2.41}$$

Let us first consider the case when $\bar{x}_j^k + \bar{d}_j^{k,\infty} = 0$ for some $j \in \mathcal{J}$. Then $r_j(x_j^k + d_j^{k,\infty}) = \|\bar{x}_j^k + \bar{d}_j^{k,\infty}\| - (x_{j0}^k + d_{j0}^{k,\infty}) = -(x_{j0}^k + d_{j0}^{k,\infty}) \leq 0$ and (2.41) holds.

Now consider the case that $\bar{x}_j^k + \bar{d}_j^{k,\infty} \neq 0$ for $j \in \mathcal{J}$. Since $d^{k,\infty}$ is a limit point of $\{d^{k,l}\}_{l=0}^\infty$, there exists a subsequence $\{d^{k,l_t}\}_{t=0}^\infty$ that converges to $d^{k,\infty}$. We may assume without loss of generality that $\bar{x}_j^k + \bar{d}_j^{k,l_t} \neq 0$ for all t. Then, for any t, by Step 30, $x_j^k + d_j^{k,l_t} \in \mathcal{Y}_j^{k,l_{t+1}}$. In the inner iteration l_{t+1}, the trial step $d_j^{k,l_{t+1}}$ is computed from (2.18) and satisfies $x_j^k + d_j^{k,l_{t+1}} \in$

$\mathcal{C}_j(\mathcal{Y}_j^{k,l_t})$, which by definition (2.14) implies

$$\nabla r_j(x_j^k + d_j^{k,l_t})^T(x_j^k + d_j^{k,l_{t+1}}) \leq 0.$$

Taking the limit $t \to \infty$ and using the fact that $\nabla r_j(v_j)^T v_j = r_j(v_j)$ for any $v_j \in \mathcal{K}_j$ yields

$$r_j(x_j^k + d_j^{k,\infty}) = \nabla r_j(x_j^k + d_j^{k,\infty})^T(x_j^k + d_j^{k,\infty}) \leq 0,$$

proving (2.41). In turn (2.41) implies that the ratio

$$\frac{\varphi(x^k + d^{k,l}; \rho^k) - \varphi(x^k; \rho^k)}{m^k(x^k + d^{k,l}; \rho^k) - m^k(x^k; \rho^k)} = \frac{c^T d^{k,l} + \rho^k([r_j(x_j^k + d_j^{k,l})]^+ - [r_j(x_j^k)]^+)}{c^T d^{k,l} - \rho^k[r_j(x_j^k)]^+}$$

converges to 1. Note that the ratio is well-defined since $m^k(x^k + d^{k,l}; \rho^k) - m^k(x^k; \rho^k) < 0$ due to Lemma 2.4.2(2). It then follows that (2.40) is true for sufficiently large l. □

Lemma 2.4.3. *Suppose that there exists $\rho^\infty > 0$ so that $\rho^k = \rho^\infty > 0$ for all large k. Then any limit point of $\{x^k\}_{k=0}^\infty$ is an optimal solution of the SOCP* (2.1).

Proof. From (2.11) and the updates in the algorithm, we have that

$$\begin{aligned}\varphi(x^{k+1}; \rho^\infty) - \varphi(x^{K_\rho}; \rho^\infty) &= \sum_{t=K_\rho}^{k} \Big(\varphi(x^{t+1}; \rho^\infty) - \varphi(x^t; \rho^\infty)\Big) \\ &\leq c_{\text{dec}} \sum_{t=K_\rho}^{k} \Big(m^t(x^t + d^t; \rho^\infty) - m^t(x^t; \rho^\infty)\Big)\end{aligned}$$

for $k \geq K_\rho$. Since the SOCP cannot be unbounded below by Assumption 2.4.1, the left-hand side is bounded below as $k \to \infty$. Lemma 2.4.1(1) shows that all summands are non-positive and we obtain

$$\lim_{k\to\infty} \Big(m^k(x^k + d^k; \rho^\infty) - m^k(x^k; \rho^\infty)\Big) = 0. \tag{2.42}$$

Using Lemma 2.4.1(1), we also have

$$\lim_{k\to\infty}\left((d^k)^T H^k d^k + (\rho^\infty - \rho^k_{\min})\sum_{j\in\mathcal{J}}[r_j(x^k_j)]^+\right) = 0.$$

Since H^k is positive semi-definite and $\rho^\infty - \rho^k_{\min} \geq \rho^\infty - \rho^\infty_{\min} > 0$, this implies that $[r_j(x^k_j)]^+ \to 0$ for all $j \in \mathcal{J}$, i.e., all limit points of $\{x^k\}^\infty_{k=0}$ are feasible. This also yields $\lim_{k\to\infty}(d^k)^T H^k d^k = 0$, and since H^k is positive semi-definite and uniformly bounded due to (2.34), we have

$$\lim_{k\to\infty} H^k d^k = 0. \tag{2.43}$$

Using (2.42) together with (2.10) and $[r_j(x^k_j)]^+ \to 0$, we obtain

$$0 = \lim_{k\to\infty}\left(c^T d^k - \rho^\infty \sum_{j\in\mathcal{D}(x^k)}[r_j(x^k_j)]^+\right) = \lim_{k\to\infty} c^T d^k. \tag{2.44}$$

Now let x^* be a limit point of $\{x^k\}^\infty_{k=0}$. Since X is bounded, d^k is bounded, and consequently there exists a subsequence $\{k_t\}^\infty_{t=0}$ of iterates so that x^{k_t} and d^{k_t} converge to x^* and d^∞, respectively, for some limit point d^∞ of d^k. Define $g^{k_t} = H^{k_t} d^{k_t}$ for all t. Then, looking at the QP optimality conditions (2.24), we see that d^{k_t} is also an optimal solution of the linear optimization problem

$$\begin{aligned} &\min_{d\in\mathbb{R}^n}\ (c + g^{k_t})^T d \\ &\text{s.t. } A(x^{k_t} + d) \leq b, \\ &\qquad x^{k_t}_j + d_j \in \mathcal{C}_j(\hat{\mathcal{Y}}^{k_t}_j), \qquad j \in \mathcal{J}. \end{aligned} \tag{2.45}$$

Now suppose, for the purpose of deriving a contradiction, that x^* is not an optimal solution of the SOCP. Since we showed above that x^* is feasible, there then exists a step

$\tilde{d}^* \in \mathbb{R}^n$ so that $\tilde{x} = x^* + \tilde{d}^*$ is feasible for (2.1) and $c^T\tilde{d}^* < 0$. Then, because $\mathcal{K}_j \subseteq \mathcal{C}_j(\hat{\mathcal{Y}}_j^{k_t})$, for each t, $\tilde{d}^{k_t} = x^* - x^{k_t} + \tilde{d}^*$ is feasible for (2.45), and because d^{k_t} is an optimal solution of (2.45), we have $(c + g^{k_t})^T d^{k_t} \leq (c + g^{k_t})^T \tilde{d}^{k_t}$. Taking the limit $t \to \infty$, we obtain $c^T d^\infty \leq c^T\tilde{d}^* < 0$, where we used $\lim_{t\to\infty} g^{k_t} = \lim_{t\to\infty} H^{k_t} d^{k_t} = 0$, due to the definition of g^{k_t} and (2.43). However, this contradicts (2.44). Therefore, x^* must be a solution of the SOCP. □

For later reference, we highlight the limit (2.43) established in the above proof.

Lemma 2.4.4. *Suppose that there exists $\rho^\infty > 0$ so that $\rho^k = \rho^\infty > 0$ for all large k. Then $\lim_{k\to\infty} H^k d^k = 0$.*

We are now ready to prove that the algorithm is globally convergent under the following standard regularity assumption.

Assumption 2.4.2. *The SOCP is feasible and Slater's constraint qualification holds, i.e., there exists a feasible point $\tilde{x} \in \mathbb{R}^n$ and $\epsilon > 0$ so that $\tilde{x} + v$ is feasible for any $v \in \mathbb{R}^n$ with $\|v\| \leq \epsilon$.*

This assumption implies that the multiplier estimates are bounded.

Lemma 2.4.5. *Suppose that Assumption 2.4.2 holds. Then $\{\hat{z}^k\}$ is bounded.*

Proof. Let $\tilde{x}$ and ϵ be the quantities from Assumption 2.4.2. Note that the QP corresponding to the optimality conditions (2.24) is

$$\begin{aligned} &\min_{d\in\mathbb{R}^n} \; c^T d + \tfrac{1}{2} d^T H^k d \\ &\text{s.t. } A(x^k + d) \leq b,\; x_j^k + d_j \in \mathcal{C}_j(\hat{\mathcal{Y}}_j^k),\; j \in \mathcal{J}. \end{aligned}$$

Since $x^{k+1} = x^k + d^k$ when d^k is the step accepted by the algorithm, it follows that x^{k+1} is an optimal solution of the QP

$$O_{\text{primal}} = \min_{x \in \mathbb{R}^n} \ (c^T - H^k x^k)x + \tfrac{1}{2} x^T H^k x$$
$$\text{s.t. } Ax^{k+1} \leq b, \ x_j^{k+1} \in \mathcal{C}_j(\hat{\mathcal{Y}}_j^k), \ j \in \mathcal{J},$$

the Lagrangian dual of which is

$$O_{\text{dual}} = \max_{x,z \in \mathbb{R}^n, \lambda \in \mathbb{R}^m} \ - b^T \lambda - \tfrac{1}{2} x^T H^k x \tag{2.46a}$$
$$\text{s.t. } c - H^k x^k + H^k x + A^T \lambda - z = 0, \tag{2.46b}$$
$$z \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k), \ j \in \mathcal{J}, \quad \lambda \geq 0. \tag{2.46c}$$

By (2.24), $(x^{k+1}, \hat{\lambda}^k, \hat{z}^k)$ is a primal-dual optimal solution of these QPs.

Define $v = -\epsilon \frac{\hat{z}^k}{\|\hat{z}^k\|}$. Then $\|v\| \leq \epsilon$, and Assumption 2.4.2 implies that $\tilde{x} + v \in \mathcal{K}_j \subseteq \mathcal{C}_j(\hat{\mathcal{Y}}_j^k)$. Since $\hat{z}^k \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k)$, we have with (2.46b) that

$$0 \leq (\tilde{x} + v)^T \hat{z}^k = v^T \hat{z}^k + \tilde{x}^T (c - H^k \tilde{x} + H^k x^{k+1} + A^T \hat{\lambda}^k). \tag{2.47}$$

Since H^k is positive definite, it is

$$0 \leq (\tilde{x} - x^{k+1})^T H^k (\tilde{x} - x^{k+1}) = \tilde{x}^T H^k \tilde{x} - 2\tilde{x}^T H^k x^{k+1} + (x^{k+1})^T H^k x^{k+1}. \tag{2.48}$$

Furthermore, Slater's condition implies strong duality, that is

$$b^T \hat{\lambda}^k + \frac{1}{2}(x^{k+1})^T H^k x^{k+1} = -O_{\text{dual}} = -O_{\text{primal}}$$
$$= -(c - H^k x^k)^T x^{k+1} - \tfrac{1}{2}(x^{k+1})^T H^k x^{k+1}. \tag{2.49}$$

Finally, since $\tilde{x}$ is feasible for the SOCP, (2.1b) and $\hat{\lambda}^k \geq 0$ imply $\tilde{x}^T A^T \hat{\lambda}^k \leq b^T \hat{\lambda}^k$. Subtracting $v^T \hat{z}^k$ on both sides of (2.47), this, together with (2.48) and (2.49), yields

$$\begin{aligned}\epsilon \|\hat{z}^k\| &\leq \tilde{x}^T c - \frac{1}{2}\tilde{x}^T H^k \tilde{x} + \frac{1}{2}(x^{k+1})^T H^k x^{k+1} + b^T \hat{\lambda}^k \\ &= \tilde{x}^T c - \frac{1}{2}\tilde{x}^T H^k \tilde{x} - c^T x^{k+1} - \frac{1}{2}(x^{k+1})^T H^k x^{k+1}.\end{aligned}$$

The first two terms are independent of k, and since X is bounded by Assumption 2.4.2 and H^k is uniformly bounded by (2.34), we can conclude that $\hat{z}^k$ is uniformly bounded. □

It is easy to see that the penalty parameter update rule (2.37) and Lemma 2.4.5 imply the following result.

Lemma 2.4.6. *Suppose Assumption 2.4.2 holds. Then there exists ρ^∞ and K_ρ so that $\rho^k = \rho^\infty > \rho^\infty_{\min}$, where $\rho^\infty_{\min} \geq \rho^k_{\min} = \|z^k_{\mathcal{J},0}\|_\infty$ for all $k \geq K_\rho$.*

We can now state the main convergence theorem of this chapter.

Theorem 2.4.1. *Suppose that Assumptions 2.4.1 and 2.4.2 hold. Then Algorithm 3 either terminates in Step 35 with an optimal solution, or it generates an infinite sequence of iterates $\{(x^{k+1}, \hat{\lambda}^k, \hat{z}^k)\}_{k=0}^\infty$, each limit point of which is a primal-dual solution of the SOCP (2.1).*

Proof. Let $\{(x^{k_t+1}, \hat{\lambda}^{k_t}, \hat{z}^{k_t})\}$ be a subsequence converging to a limit point (x^*, λ^*, z^*). No matter whether an iterate is computed from the optimal solution of (2.18), (2.27), or (2.35), the iterates satisfy the optimality conditions (2.24). In particular, from (2.24c) we have for any $j \in \mathcal{J}$ that $\hat{z}_j^{k_t} \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^{k_t}) \subseteq \mathcal{K}_j$ and $(x_j^{k_t+1})^T \hat{z}_j^{k_t} = 0$. In the limit, we obtain $z_j^* \in \mathcal{K}_j$ (since $\mathcal{K}_j$ is closed) and $(x_j^*)^T z_j^* = 0$. Lemma 2.4.6 yields that $\rho^k = \rho^\infty$ for all large k, and so Lemma 2.4.3 implies that x^* is feasible, i.e., $x_j^* \in \mathcal{K}_j$. Therefore, (2.13c)

holds. Using Lemma 2.4.4 we can take the limit in (2.24a) and (2.24b) and deduce also the remaining SOCP optimality conditions (2.13a) and (2.13b) hold at the limit point. □

Remark 2.4.1. *In case the SOCP is infeasible, we have two possible outcomes. Either, Algorithm 3 terminates in some iterations because one of the QPs is infeasible, or* $\lim_{k\to\infty} \rho^k = \infty$ *(reverse conclusion of Lemma 2.4.3).*

2.4.2. Identification of extremal-active cones

We can only expect fast local convergence under some non-degeneracy assumptions. Throughout this chapter, we assume that Assumption 2.2.2 holds. Under this assumption, (x^*, λ^*, z^*) is the unique optimal solution [**2**, Theorem 22], and Theorem 2.4.1 then implies that

$$\lim_{k\to\infty} (x^{k+1}, \hat{\lambda}^k, \hat{z}^k) = (x^*, \lambda^*, z^*).$$

First, we prove a technical result that describes elements in $\mathcal{C}_j^\circ(\mathcal{Y}_j)$ in a compact manner. For this characterization to hold, condition (2.20) for the initialization $\mathcal{Y}_j^0$ of the set of hyperplane-generating points is crucial.

Lemma 2.4.7. *Let* $y_j \in \mathbb{R}^{n_j}$ *with* $\bar{y}_j \neq 0$ *and* $y_{j0} \geq 0$. *Further, let* $\Phi_j(z_j, y_j) := z_{j0} - \|\bar{z}_j + \bar{y}_j\|_1 - \|\bar{y}_j\|$. *Then the following statements hold for* $z_j, y_j \in \mathbb{R}^{n_j}$*:*

(1) $z_j \in \mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\})$ *if* $\Phi_j(z_j, y_j) \geq 0$.

(2) $z_j \in \operatorname{int}(\mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\}))$ *if* $\Phi_j(z_j, y_j) > 0$.

Proof. For (1): Suppose $\Phi_j(z_j, y_j) \geq 0$, then

$$z_{j0} \geq \|\bar{z}_j + \bar{y}_j\|_1 + \|\bar{y}_j\|.$$

Define $\bar{s}_j = \bar{z}_j + \bar{y}_j$ and choose $\sigma_j^+ \in \mathbb{R}_+^{n_j-1}$ and $\sigma_j^- \in \mathbb{R}_+^{n_j-1}$ so that $\bar{s}_j = \sigma_j^+ - \sigma_j^-$ and $|s_{ji}| = \sigma_{ji}^+ + \sigma_{ji}^-$ for all $i = 1, \ldots, n_j - 1$. Then we have

$$z_{j0} = \sum_{i=1}^{n_j-1} \sigma_{ji}^+ + \sum_{i=1}^{n_j-1} \sigma_{ji}^- + \sigma_j + \eta_j$$

$$\bar{z}_j = \bar{s}_j - \bar{y}_j = \sigma_j^+ - \sigma_j^- - \sigma_j \frac{\bar{y}_j}{\|\bar{y}_j\|}$$

with $\sigma_j = \|\bar{y}_j\|$ and some $\eta_j \in \mathbb{R}_+$. Using (2.15), this can be rewritten as

$$z_j = -\sum_{i=1}^{n_j-1} \sigma_{ji}^+ \nabla r_j(-e_{ji}) - \sum_{i=1}^{n_j-1} \sigma_{ji}^- \nabla r_j(e_{ji}) - \sigma_j \nabla r_j(y_j) + \eta_j e_{j0}.$$

By the definition of $\mathcal{C}_j^\circ$ in (2.17), this implies that $z_j \in \mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^0 \cup \{y_j\})$ where $\hat{\mathcal{Y}}_j^0$ is defined in (2.20). Since $\hat{\mathcal{Y}}_j^0 \subseteq \mathcal{Y}_j^0$ from (2.20), we have $\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^0 \cup \{y_j\}) \subseteq \mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\})$, and the claim follows.

For (2): Suppose $\Phi_j(z_j, y_j) > 0$. Because Φ_j is a continuous function, there exists a neighborhood $N_\epsilon(z_j)$ around z_j so that $\Phi_j(\hat{z}_j, y_j) > 0$ for all $\hat{z}_j \in N_\epsilon(z_j)$. From part (1) we then have $N_\epsilon(z_j) \subseteq \mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\})$, and consequently $z_j \in \text{int}(\mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\}))$. □

Theorem 2.4.2. *For all k sufficiently large, we have $\mathcal{E}(x^k) = \mathcal{E}(x^*)$.*

Proof. Choose $j \notin \mathcal{E}(x^*)$, then $x_j^* \neq 0$. Because $x_j^k \to x_j^*$, it is $x_j^k \neq 0$ or, equivalently, $j \notin \mathcal{E}(x^k)$ for k sufficiently large. For the remainder of this proof we consider $j \in \mathcal{E}(x^*)$ and show that $j \in \mathcal{E}(x^k)$ for large k. Note that strict complementarity in Assumption 2.2.2 implies that $z_j^* \in \text{int}(\mathcal{K}_j)$, i.e., $r_j(z_j^*) < 0$, and consequently $z_{j0}^* > 0$.

First consider the iterations in which fast NLP-SQP steps are accepted in Steps 11 or 16. For the purpose of deriving a contradiction, suppose there exists an infinite subsequence so that $x^{k_t+1} = x^{k_t} + d^{S,k_t}$ or $x^{k_t+1} = x^{k_t} + d^{S,k_t} + s^{k_t}$ and $j \notin \hat{\mathcal{E}}^{k_t}$. Then $j \notin \hat{\mathcal{E}}^{k_t}$ implies

$x_{j0}^{k_t+1} > 0$ (according to the termination condition in the while loop in Step 6). We also have $\hat{\mathcal{Y}}_j^{k_t} = \{\check{x}_j^{k_t}\}$ where $\check{x}_j^{k_t} = x_j^{k_t}$ from (2.28) or $\check{x}_j^{k_t} = x_j^{k_t} + d_j^{S,k_t}$ from (2.36). Condition (2.24c) yields $z_j^{k_t} \in \mathcal{C}_j^\circ(\{\check{x}_j^{k_t}\})$, so by (2.17) it is $z_j^{k_t} = -\sigma_j \nabla r_j(\check{x}_j^{k_t}) + \eta_j e_{j0}$ for some $\sigma_j, \eta_j \geq 0$, as well as $x_j^{k_t+1} \in \mathcal{C}_j(\{\check{x}_j^{k_t}\})$, which by (2.14) implies $\nabla r_j(\check{x}_j^{k_t})^T x_j^{k_t+1} \leq 0$. Then complementarity yields

$$0 = (z_j^{k_t})^T x_j^{k_t+1} = -\sigma_j \nabla r_j(\check{x}_j^{k_t})^T x_j^{k_t+1} + \eta_j x_{j0}^{k_t+1} \geq \eta_j x_{j0}^{k_t+1}.$$

Since $x_{j0}^{k_t+1} > 0$ and $\eta_j \geq 0$, we must have $\eta_j = 0$, and consequently $z_j^{k_t} = -\sigma_j \nabla r_j(\check{x}_j^{k_t})$. It is easy to see that $r_j(-\sigma_j \nabla r_j(\check{x}_j^{k_t})) = 0$. Since $z_j^{k_t} \to z_j^*$, continuity of r_j yields $r_j(z_j^*) = 0$, in contradiction to $z_j^* \in \text{int}(\mathcal{K}_j)$. We thus showed that $j \in \hat{\mathcal{E}}^k$ for all large iterations k in which the NLP-SQP step was accepted, and consequently (2.28) and (2.36) yield $\hat{\mathcal{Y}}_j^k = \mathcal{Y}_j^k$ for such k.

In all other iterations (2.22) holds, and overall we obtain

$$\mathcal{Y}_j^0 \subseteq \mathcal{Y}_j^k \subseteq \hat{\mathcal{Y}}_j^k \text{ for all sufficiently large } k. \tag{2.50}$$

Let us first consider the case when $\bar{z}_j^* = 0$. Then $\|\bar{z}_j^*\| - z_{j0}^* = r_j(z_j^*) < 0$ yields $z_{j0}^* > 0$. To apply Lemma 2.4.7 choose any $i \in \{1, \ldots, n_j - 1\}$ and let $y_j = e_{ji}$. Then $\|y_j\|_1 = \|y_j\| = 1$ and $\Phi_j(z_j^*, y_j) = z_{j0}^* > 0$. Since $\hat{z}_j^k \to z_j^*$ and Φ_j is continuous, $\Phi_j(\hat{z}_j^k, y_j) > 0$ for sufficiently large k, and by Lemma 2.4.7, $\hat{z}_j^k \in \text{int}(\mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{y_j\}))$. Since $y_j \in \mathcal{Y}_j^0$ and (2.50) holds, we also have $\hat{z}_j^k \in \text{int}(\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^k))$. General conic complementarity in (2.24c) then implies that $x_j^{k+1} = x_j^k + d_j^k = 0$ for all large k, or equivalently, $j \in \mathcal{E}(x^k)$ for k sufficiently large, as desired.

Now consider the case $\bar{z}_j^* \neq 0$. For the purpose of deriving a contradiction, suppose there exists a subsequence $\{x^{k_t}\}_{t=0}^\infty$ so that $j \notin \mathcal{E}(x^{k_t})$, i.e., $x^{k_t} \neq 0$, for all t. Because $\check{z}_j^k \to z_j^*$,

$\bar{z}_j^* \neq 0$, and $r_j(z_j^*) < 0$, we may assume without loss of generality that $r_j(\check{z}_j^{k_t}) < 0$ and $\bar{\check{z}}_j^{k_t} \neq 0$ for all t. Using this and $x_j^{k_t} \neq 0$, we see that the update rule (2.26) in Step 33 adds $-\check{z}_j^{k_t}$ to $\mathcal{Y}_j^{k_t+1}$. With (2.50), we have

$$-\check{z}_j^{k_t} \in \mathcal{Y}_j^{k_t+1} \subseteq \mathcal{Y}_j^{k_{t+1}} \subseteq \hat{\mathcal{Y}}_j^{k_{t+1}} \text{ for all } t. \tag{2.51}$$

Recall the mapping Φ_j defined in Lemma 2.4.7 and note that $\Phi_j(z_j^*, -z_j^*) = z_{j0}^* - \|\bar{z}_j^*\| = -r_j(z_j^*) > 0$. Since both $\hat{z}_j^k$ and $\check{z}_j^k$ converge to z_j^* and Φ_j is continuous, it is $\Phi_j(\hat{z}_j^{k_{t+1}-1}, -\check{z}_j^{k_t}) > 0$ for all large t, and therefore, by Lemma 2.4.7, $\hat{z}_j^{k_{t+1}-1} \in \operatorname{int}(\mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{-\check{z}_j^{k_t}\})) \overset{(2.51)}{\subseteq} \operatorname{int}(\mathcal{C}_j^\circ(\hat{\mathcal{Y}}_j^{k_{t+1}-1}))$ for all large t. Conic complementarity in (2.24c) then implies that $x_j^{k_{t+1}} = x_j^{k_{t+1}-1} + d_j^{k_{t+1}-1} = 0$. This is a contradiction of the definition of the subsequence $\{x^{k_t}\}_{t=0}^\infty$. □

Remark 2.4.2. *In the proof of Theorem 2.4.2, we saw that $\Phi_j(z_j^*, -z_j^*) > 0$ if $j \in \mathcal{E}(x^*)$ and $\bar{z}_j^* \neq 0$. Since Φ_j is continuous, this implies that there exists a neighborhood $N_\epsilon(z_j^*)$ so that $\Phi_j(z_j, -y_j) > 0$, and consequently $z_j \in \operatorname{int}(\mathcal{C}_j^\circ(\mathcal{Y}_j^0 \cup \{-y_j\}))$, for all $z_j, y_j \in N_\epsilon(z_j^*)$.*

2.4.3. Quadratic local convergence

As discussed in Chapter 2.2.1, since x^* is a solution of the SOCP (2.1), it is also a solution of the nonlinear problem (2.4). We now show that Algorithm 3 eventually generates steps that are identical to SQP steps for (2.4). Then Theorem 2.2.2 implies that the iterates converge locally at a quadratic rate.

We first need to establish that the assumptions for Theorem 2.2.2 hold.

Lemma 2.4.8. *Suppose that Assumption 2.2.2 holds for the SOCP* (2.1)*. Then Assumption 2.2.1 holds for the NLP* (2.4)*.*

Proof. Let λ^* and z^* be the optimal multipliers for the SOCP corresponding to x^*, satisfying (2.13). Assumption 2.2.2 implies that λ^* and z^* are unique [**2**, Theorem 22].

Let $j \in \mathcal{D}(x^*)$ and define $\mu_j^* = z_{j0}^* \geq 0$. If $0 = r_j(x_j^*) = x_{j0}^* - \|\bar{x}_j^*\|$, complementarity (2.13c) implies, for all $i \in \{1, \dots n_j\}$, that $0 = x_{j0}^* z_{ji}^* + x_{ji}^* z_{j0}^* = \|\bar{x}_j^*\| z_{ji}^* + x_{ji}^* z_{j0}^*$,, or equivalently, $z_{ji}^* = -z_{j0}^* \frac{x_{ji}^*}{\|\bar{x}_j^*\|}$; see [**2**, Lemma 15]. Using (2.15), this can be written as

$$z_j^* = -z_{j0}^* \nabla r_j(x_j^*) = -\mu_j^* \nabla r_j(x_j^*). \tag{2.52}$$

On the other hand, if $r_j(x_j^*) < 0$, i.e., the constraint (2.4c) is inactive, then $x_j^* \in \text{int}(\mathcal{K}_j)$ and complementarity (2.13c) yields $z_j^* = 0$ (see [**2**, Definition 23]) and therefore $\mu_j^* = 0$. Consequently, (2.52) is also valid in that case. Finally, we define $\nu_j^* = z_j^*$ for all $i \in \mathcal{E}(x^*)$. With these definitions, (2.13a) can be restated as

$$c + A^T \lambda^* + \sum_{j \in \mathcal{D}(x^*)} \mu_j^* \nabla r_j(x^*) - \nu^* = 0, \tag{2.53}$$

where $\nu^* \in \mathbb{R}^n$ is the vector with the values of ν_j^* at the components corresponding to $j \in \mathcal{E}(x^*)$ and zero otherwise. We now prove parts (1), (2), and (3) of Assumption 2.2.1.

Proof of (1): Let $j \in \mathcal{D}(x_j^*)$. We already established that $r_j(x_j^*) < 0$ yields $\mu_j^* = 0$. Now suppose that $r_j(x_j^*) = 0$. Then $x_j^* \in \text{bd}(\mathcal{K}_j) \setminus \{0\}$. Since strict complementarity is assumed, we have $z_j^* \in \text{bd}(\mathcal{K}_j) \setminus \{0\}$ (see the comment after Assumption 2.2.2), which in turn yields $z_j^* \neq 0$ and hence $\mu_j^* \neq 0$.

Proof of (2): Since we need to prove linear independence only of those constraints that are active at x^*, we consider only those rows $A_{\mathcal{A}}$ of A for which (2.4b) is binding.

Without loss of generality suppose x^* is partitioned into four parts, $(x^*)^T = ((x_{\mathcal{B}}^*)^T \ (x_{\mathcal{I}}^*)^T \ (x_{\mathcal{E}}^*)^T \ (x_{\mathcal{F}}^*)^T)$, where $x_{\mathcal{B}}^*$, $x_{\mathcal{I}}^*$, and $x_{\mathcal{E}}^*$ correspond to the variables in the cones $\mathcal{B} = \{j \in$

$\mathcal{J} : r_j(x_j^*) = 0, x_j^* \neq 0\}$, $\mathcal{I} = \{j \in \mathcal{J} : r_j(x_j^*) < 0\}$, and $\mathcal{E} = \mathcal{E}(x^*)$, respectively, and $x_{\mathcal{F}}^*$ includes all components of x^* that are not in any of the cones. Further suppose that $(x_{\mathcal{B}}^*)^T = ((x_1^*)^T \ \ldots (x_{p_{\mathcal{B}}}^*)^T)$, where $\mathcal{B} = \{1, \ldots, p_{\mathcal{B}}\}$, and that $A_{\mathcal{A}}$ is partitioned in the same way.

Primal non-degeneracy of the SOCP implies all that matrices of the form

$$\begin{pmatrix} [A_{\mathcal{A}}]_1 & \cdots & [A_{\mathcal{A}}]_{p_{\mathcal{B}}} & [A_{\mathcal{A}}]_{\mathcal{I}} & [A_{\mathcal{A}}]_{\mathcal{E}} & [A_{\mathcal{A}}]_{\mathcal{F}} \\ \alpha_1 \nabla r_1(x_1^*)^T & \cdots & \alpha_{p_{\mathcal{B}}} \nabla r_{p_{\mathcal{B}}}(x_{p_{\mathcal{B}}}^*)^T & 0^T & v^T & 0^T \end{pmatrix}$$

have linear independent rows for all scalars α_i and vectors v, not all zero [**2**, Eq. (50)]. This implies that the rows of $A_{\mathcal{A}}$, together with the gradient of any one of the binding constraints in (2.4c) and (2.4d) are linearly independent. Because the constraint gradients, which are of the form $\nabla r_j(x_j^*)$ and e_{ij}, share no nonzero components when extended to the full space, we conclude that the gradients of all active constraints are linearly independent at x^*, i.e., the LICQ holds.

Proof of (3): For the purpose of deriving a contradiction, suppose that there exists a direction $d \in \mathbb{R}^n \setminus \{0\}$ that lies in the null space of the constraints of (2.4) that are binding at x^* and for which $d^T H^* d \leq 0$.

Since d is in the null space of the binding constraints, we have $A_{\mathcal{A}} d = 0$, $\nabla r_j(x^*)^T d = 0$ for $j \in \mathcal{B}$, and $d_j = 0$ for all $j \in \mathcal{E}$. Premultiplying (2.53) by d^T gives

$$0 = c^T d + (\lambda^*)^T \underbrace{A_{\mathcal{A}} d}_{0} + \sum_{j \in \mathcal{B}} \mu^* \underbrace{\nabla r_j(x^*)^T d}_{0} + \sum_{j \in \mathcal{I}} \underbrace{\mu^*}_{0} \nabla r_j(x^*)^T d + \underbrace{(\nu^*)^T d}_{0} = c^T d. \tag{2.54}$$

What remains to show is that d is a feasible direction for the SOCP, i.e., there exists $\beta > 0$ so that $x^* + \beta d$ is feasible for the SOCP. Because of (2.54), this point has the same objective value as x^* and is therefore also an optimal solution of the SOCP. This contradicts the fact that Assumption 2.2.2 implies that the optimal solution is unique [**2**, Theorem 22].

By the definition of H^* in Assumption 2.2.1 and the choice of d, we have

$$0 \geq d^T H^* d = \sum_{j \in \mathcal{D}(x^*)} \mu_j^* d_j^T \nabla^2 r_j(x_j^*) d_j = \sum_{j \in \mathcal{B}} \mu_j^* d_j^T \nabla^2 r_j(x_j^*) d_j.$$

Since for all $j \in \mathcal{B}$, the Hessian $\nabla^2 r_j(x_j^*)$ is positive semi-definite and $\mu_j^* > 0$ from Part (i), this yields $d_j^T \nabla^2 r_j(x_j^*) d_j = 0$ for all $j \in \mathcal{B}$.

Let $j \in \mathcal{B}$. Then from (2.7)

$$0 = d_j^T \nabla^2 r_j(x^*) d_j = \frac{\|\bar{d}_j\|^2 \|\bar{x}_j^*\|^2 - (\bar{d}_j^T \bar{x}_j^*)^2}{\|\bar{x}_j^*\|^3}. \tag{2.55}$$

The definition of $\mathcal{B}$ implies $r_j(x_j^*) = 0$ and so $x_{j0}^* = \|\bar{x}_j^*\|$. Since d_j is in the null space of $\nabla r_j(x_j^*)$, we have $0 = \nabla r_j(x_j^*)^T d_j = -d_{j0} + \frac{\bar{d}_j^T \bar{x}_j}{\|\bar{x}_j^*\|}$, which in turn yields $d_{j0} x_{j0}^* = \bar{d}_j^T \bar{x}_j^*$. Finally, using these relationships together with (2.55) gives

$$0 = \|\bar{d}_j\|^2 \|\bar{x}_j^*\|^2 - (\bar{d}_j^T \bar{x}_j^*)^2 = \|\bar{d}_j\|^2 (x_{j0}^*)^2 - (d_{j0} x_{j0}^*)^2$$

and so $d_{j0}^2 = \|\bar{d}_j\|^2$. All of these facts imply that for any $\beta \in \mathbb{R}$,

$$\begin{aligned} &\|\bar{x}_j^* + \beta \bar{d}_j\|^2 - (x_{j0}^* + \beta d_{j0})^2 \\ =&\|\bar{x}_j^*\|^2 + 2\beta \bar{d}_j^T \bar{x}_j^* + \beta^2 \|\bar{d}_j\|^2 - \left((x_{j0}^*)^2 + 2\beta d_{j0} x_{j0}^* + \beta^2 d_{j0}^2\right) = 0, \end{aligned}$$

which implies $r_j(x_j^* + \beta d_j) = 0$ and therefore $x_j^* + \beta d_j \in \mathcal{K}_j$.

Further, because d lies in the null space of the active constraints, we have, for any $\beta \in \mathbb{R}$, that $x_j^* + \beta d_j = 0 \in \mathcal{K}_j$ for all $j \in \mathcal{E}(x^*)$ and $A_{\mathcal{A}}(x^* + \beta d) = b_{\mathcal{A}}$. Finally, since $r_j(x_j^*) < 0$ and hence $x_j^* \in \text{int}(\mathcal{K}_j)$ for all $j \in \mathcal{J} \setminus (\mathcal{E}(x^*) \cup \mathcal{B})$, and since x_j^* is strictly feasible for all non-binding constraints in (2.1b), there exists $\beta > 0$ so that $x^* + \beta d$ satisfies all constraints in (2.1). □

Theorem 2.4.3. *Suppose that $c_H > \|H^*\|$. Then the primal-dual iterates $(x^{k+1}, \hat{\lambda}^k, \hat{z}^k)$ converge locally to (x^*, λ^*, z^*) at a quadratic rate.*

Proof. We already established in Theorem 2.4.1 that the iterates converge to the optimal solution, and since $H^k \to H^*$ and $c_H > \|H^*\|$, the Hessian is not rescaled according to (2.34) in Step 3. Using Theorem 2.4.2 we know that, once k is sufficiently large, the step $d^{S,k}$ computed in Step 5 of Algorithm 3 is identical with the SQP step from (2.5) for (2.4); we can ignore (2.27d) here because $x_{j0}^* > 0$ and $d_{j0}^{S,k} \to 0$ and therefore this constraints is not active for large k. This also implies that the condition in Step 6 is never true and thus $\hat{\mathcal{E}}_k = \mathcal{E}(x^*)$. If the decrease condition in Step 11 is not satisfied, by a similar argument we have that s^k computed in Step 15 is the second-order correction step from (2.12) for (2.4). Due to Lemma 2.4.8 we can now apply Theorem 2.2.2 to conclude that either $d^{S,k}$ or $d^{S,k} + s^k$ is accepted to define the next iterate for large k and that the iterates converge at a quadratic rate. □

2.5. Numerical Experiments

In this chapter, we examine the performance of Algorithm 3. First, using randomly generated instances, we consider three types of starting points: (i) uninformative default starting point (cold start), (ii) solution of a perturbed instance, (iii) solution computed by an interior-point SOCP solver whose accuracy we wish to improve. Then we briefly report

results using the test library CBLIB. The numerical experiments were performed on an Ubuntu 22.04 Linux server with a 2.1GHz Xeon Gold 5128 R CPU and 256GB of RAM.

2.5.1. Implementation

We implemented Algorithm 3 in MATLAB R2021b, with parameters $c_{\text{dec}} = 10^{-6}$, $c_{\text{inc}} = 2$, $c_H = 10^{12}$, and $\rho^{-1} = 50$. In each iteration, we identify $\mathcal{E}(x^k) = \{j \in \mathcal{J} : \|x_j^k\|_\infty < 10^{-6}\}$ and $\mathcal{D}(x^k) = \{j \in \mathcal{J} \setminus \mathcal{E}(x^k) : \|\bar{x}_j^k\| > 10^{-8}\}$. The set $\mathcal{Y}_j^0$ is initialized to $\hat{\mathcal{Y}}_j^0$ (see (2.20)), and λ^0 is a given starting value for λ, if provided, and zero otherwise. In addition, since the identification of the optimal extremal-active set $\mathcal{E}(x^*)$ requires $z_j^* \in \mathcal{C}_j^\circ(\mathcal{Y}_j)$, we add $-\check{z}_j^0$ to $\mathcal{Y}_j^0$, where $\check{z}^0 = c + A^T\lambda^0$.

The algorithm terminates when the violation of the SOCP optimality conditions (2.13) for the current iterate satisfies

$$E(x^k, \lambda^k, \check{z}^k) = \max \left\{ \begin{array}{l} \|[Ax^k - b]^+\|_\infty, \|(Ax^k - b) \circ \lambda^k\|_\infty, \|[-\lambda^k]^+\|_\infty \\ \\ \max_{j\in\mathcal{J}} \left\{ [r_j(x^k)]^+, [r_j(\check{z}^k)]^+, |(x_j^k)^T \check{z}_j^k| \right\} \end{array} \right\} \leq \epsilon_{\text{tol}} \tag{2.56}$$

with $\check{z}^k = c + A^T\lambda^k$, for some $\epsilon_{\text{tol}} > 0$.

As in [**81**], the sufficient descent condition (2.11) is slightly relaxed by

$$\varphi(\hat{x}^{k+1}; \rho^k) - \varphi(x^k; \rho^k) - 10\epsilon_{\text{mach}}|\varphi(x^k; \rho^k)| \leq c_{\text{dec}} \left(m^k(x^k + d; \rho^k) - m^k(x^k; \rho^k) \right)$$

to account for cancellation error, where ϵ_{mach} is the machine precision. Finally, to avoid accumulating very similar hyperplanes that would lead to degenerate QPs, we do not add a new generating point v_j to $\mathcal{Y}_j^k$ if there already exists $y_j \in \mathcal{Y}_j^k$ such that $\left\| \frac{\bar{v}_j}{\|\bar{v}_j\|} - \frac{\bar{y}_j}{\|\bar{y}_j\|} \right\|_\infty \leq 10^{-10}$.

In these experiments, we disabled the second-order correction step (Steps 15–19) because we noticed that it was never accepted in practice. In a more sophisticated implementation, one would include a heuristic that attempts to detect the Maratos effect and then triggers the second-order correction step in specific situations.

The QPs were solved using ILOG CPLEX V12.10, with optimality and feasibility tolerances set to 10^{-9} and "dependency checker" and "numerical precision emphasis" enabled, using the primal simplex method. When CPLEX did not report a solution status "optimal" and the QP KKT error was above 10^{-9}, a small perturbation was added to the Hessian matrix, i.e., we replaced H^k by $H^k + 10^{-7} \cdot I$. This helped in some cases in which CPLEX (incorrectly) reported that H^k was not positive semi-definite. If CPLEX still did not find a QP solution with KKT error less than 10^{-9}, we attempted to resolve the QP with the barrier method, the dual simplex method, and the primal simplex method again, until one was able to compute a solution. If all solvers failed for QP (2.27), the algorithm continued in Step 21. If no solver was able to solve (2.18), we terminated the main algorithm and declared a failure.

We emphasize that the purpose of our implementation is to assess whether the proposed algorithm exhibits behavior that validates the stated goals: Convergence from any starting point and rapid local convergence to highly accurate solutions. In its current implementation, it requires more computation time than highly sophisticated commercial solvers such as MOSEK or CPLEX, which were developed over decades and have highly specialized linear algebra routines that are tightly integrated into the algorithms. As we observed at the end of Chapter 2.3.7, many of the QPs in Algorithm 3 that are solved in succession are similar to each other, and savings in computation times should therefore be achievable. However, our prototype implementation based on the Matlab CPLEX interface does not allow us to

n	m	K	solved	total iter	SQP iter	total QP (2.27)	total QP (2.18)
200	60	10	30	6.67	6.67	9.77	0.00
400	120	20	30	7.20	7.20	11.57	0.00
1000	300	50	30	7.23	7.23	12.17	0.00
200	60	4	30	7.53	7.07	11.83	0.90
400	120	8	30	8.27	7.77	14.20	1.00
1000	300	20	30	8.67	7.80	15.93	1.83
200	60	2	30	8.47	7.87	13.90	1.20
400	120	4	30	8.87	8.07	15.30	1.60
1000	300	10	30	9.47	8.43	17.27	1.97

Table 2.1. Results with $x^0 = 0$, $\epsilon_{\text{tol}} = 10^{-7}$, average per-size statistics taken over 30 random instances. "solved": number of instances solved (out of 30); "total iter": total number of iterations in Algorithm 3; "SQP iter": number of iteration in which NLP-SQP step was accepted in Steps 11 or 16; "total QP (2.27)" / "total QP (2.18)": Total number of QPs of that type solved.

utilize callback functions for adding or removing hyperplanes. Achieving these savings in computation time thus requires a more sophisticated implementation, a task that is outside of the scope of this paper. Consequently, we do not report solution times here.

2.5.2. Randomly generated QCQPs

The experiments were performed on randomly generated SOCP instances of varying sizes, specified by (n, m, K). Here, $n, m \geq 1$ are the number of variables and linear constraints, respectively. $K \geq 1$ specifies the number of cones of each "activity type": $|\mathcal{E}(x^*)| = K$, $|\{j \in \mathcal{J} : r_j(x_j^*) = 0, x_j^* \neq 0\}| = K$, and $|\{j \in \mathcal{J} : r_j(x_j^*) < 0\}| = K$, i.e., there are K cones that are extremal-active, K that are active at the boundary, and K that are inactive at the optimal solution x^*. The dimensions of the cones are randomly chosen. In addition, there are variables that are not part of any cone, with bounds chosen in a way so that the non-degeneracy assumption, Assumption 2.2.2, holds. A detailed description of the problem generation is stated in Appendix A.

Table 2.1 summarizes the performance of the algorithm with an uninformative starting point $x^0 = 0$. Each row lists average statistics for a given problem size (n, m, K), taken over 30 random instances. We see that the proposed algorithm is very reliable and solved every instance to the tolerance $\epsilon = 10^{-7}$. The average number of iterations is mostly between 7–9, during most of which the second-order NLP-SQP step was accepted.

To give an idea of the computational effort, we report the number of times QPs (2.27) and (2.18) were solved. And we can draw further conclusions from this data: Consider, for example, the last row. At the beginning of each iteration, QP (2.27) is solved to obtain the NLP-SQP step. The difference with the total number of iterations, i.e., 17.27-9.47=7.80, gives us the total number of times in which the guess $\hat{\mathcal{E}}^k$ of the extremal-active cones needed to be corrected in Steps 6–9. In other words, on average, the loop Steps 6–9 is executed 7.80/9.47=0.82 times per iteration. Similarly, the last column tells us the total number of iterations of the loop in Steps 21–31. The loop was only executed when the NLP-SQP step was not accepted, so in 9.47-8.43=1.04 iterations, taking 1.97/1.04=1.89 loop iterations on average.

The experiments are presented in three groups where the ratio between n and K is kept constant. As the number of cones, K, decreases from one group to the next, the average size of the individual cones increases by a factor of 2.5 and 2, respectively. This increase seems to result in slightly more iterations in which the SQP step was rejected, indicating that the simple linearization (2.27c) of the non-extremal-active cones becomes sometimes insufficiently accurate.

The remaining experiments in this chapter investigate to which degree the algorithm is able to achieve our primary goal of taking advantage of a good starting point. We begin with an extreme situation, in which we first solve an instance with the interior-point SOCP solver

n	m	K	total iter	SQP iter	total QP (2.27)	total QP (2.18)	Mosek error	final error
200	60	10	1.10	1.07	1.10	0.07	2.33e-06	1.63e-10
400	120	20	1.03	1.00	1.03	0.03	2.67e-06	1.70e-10
1000	300	50	1.07	1.03	1.07	0.03	3.49e-06	1.76e-10
200	60	4	1.03	1.03	1.03	0.00	5.97e-06	1.69e-10
400	120	8	1.00	1.00	1.00	0.00	2.28e-06	1.87e-10
1000	300	20	1.03	0.83	1.03	0.27	5.20e-06	1.72e-10
200	60	2	1.00	1.00	1.00	0.00	2.02e-06	1.53e-10
400	120	4	1.13	1.10	1.13	0.03	4.85e-06	2.03e-10
1000	300	10	1.20	1.10	1.20	0.13	1.22e-05	2.41e-10

Table 2.2. Result with MOSEK solution as x^0, $\epsilon_{\text{tol}} = 10^{-9}$. All instances were solved. "Mosek error": Optimality error E (2.56) at Mosek solution; "final error": Optimality error E at final iterate of Algorithm 3.

n	m	K	solved	total iter	SQP iter	total QP (2.27)	total QP (2.18)
200	60	10	30	1.00	0.97	1.00	0.07
400	120	20	30	1.00	0.97	1.00	0.03
1000	300	50	30	1.00	0.97	1.00	0.07
200	60	4	30	1.00	1.00	1.00	0.00
400	120	8	30	1.00	0.93	1.00	0.07
1000	300	20	30	1.00	0.87	1.00	0.20
200	60	2	30	1.00	1.00	1.00	0.00
400	120	4	30	1.00	0.97	1.00	0.03
1000	300	10	30	1.07	1.00	1.03	0.07

Table 2.3. Result with 10^{-3} perturbation, $\epsilon_{\text{tol}} = 10^{-7}$.

MOSEK V9.1.9 (called via CVX), using the setting `cvx_precision=high` corresponding to the MOSEK tolerance $\epsilon = \epsilon_{\text{mach}}^{2/3}$, and give the resulting primal-dual solution as starting point to Algorithm 3. Choosing any tighter MOSEK tolerances leads to failures in several problems. Table 2.2 summarizes the results. In all cases, the algorithm converges rapidly to an improved solution, reducing the error by 4 orders of magnitude, most of the time with only a single second-order iteration. The Mosek error was dominated by the violation of complementarity. This demonstrates the ability of the proposed method to improve the accuracy of a solution computed by an interior-point method.

n	m	K	solved	total iter	SQP iter	total QP (2.27)	total QP (2.18)
200	60	10	30	1.20	1.07	1.00	0.19
400	120	20	30	1.33	1.17	1.00	0.73
1000	300	50	30	1.60	1.23	1.02	1.29
200	60	4	30	1.27	1.13	1.02	0.71
400	120	8	30	1.67	1.27	1.16	0.48
1000	300	20	30	2.10	1.40	1.27	0.57
200	60	2	30	1.67	1.33	1.24	0.42
400	120	4	30	2.30	1.87	1.30	0.38
1000	300	10	30	3.67	2.53	1.53	0.59

Table 2.4. Result with 10^{-1} perturbation, $\epsilon_{\text{tol}} = 10^{-7}$.

For the final experiments, summarized in Tables 2.3 and 2.4, the starting point is the MOSEK solution of a perturbed problem, in which 10% of the objective coefficients c were perturbed by uniformly distributed random noise of the order of 10^{-3} and 10^{-1}, respectively. For the small perturbation, similar to Table 2.2, Algorithm 3 terminated in one iteration most of the time. More iterations were required for the larger perturbation, but still significantly fewer compared to the uninformative starting point, see Table 2.1.

2.5.3. CBLIB instances

To demonstrate the robustness of the algorithm we also solved instances from the Conic Benchmark Library CBLIB [**31**]. Some instances involve rotated second-order cone constraints, and we reformulated them so that they fit into our standard form (2.1). We chose all 1,575 instances with at most 10,000 variables and 10,000 constraints. Integer variables were relaxed to be continuous.

Table 2.5 shows the statistics for the starting point $x^0 = 0$, where instances were grouped into sets of problems with similar names, and the remaining ones are collected in `misc`. The method was able to solve 99.2% of the instances, where 10 problems could not be solved due

Problem subset	# var	# con	# soc	solved/ total	total iter	SQP iter	iter warm/ iter cold
10/20*	3024.0	6030.6	336.0	10/11	6.5	6.5	0.57
achtziger_*	2314.0	1856.7	257.0	6/7	21.3	10.2	0.53
as_*	4909.0	5695.7	669.0	20/20	10.4	8.4	0.37
ck_*	2376.0	2375.0	15.0	90/90	5.8	4.6	0.40
classical_*	159.3	200.4	1.0	409/409	7.3	7.3	0.49
clay*	457.2	691.9	82.9	11/12	67.6	13.2	1.31
estein*	103.1	165.2	14.0	9/9	18.3	4.4	0.30
flay*	175.0	408.0	4.0	10/10	7.0	3.6	0.34
fo[7-9]*	208.6	540.2	15.9	19/19	12.3	3.0	0.53
m[3-9]*	150.2	370.5	12.8	8/8	10.9	4.0	0.44
nb*	3098.8	321.0	816.0	4/4	4.8	4.2	3.43
netmod*	989.7	2593.0	4.7	3/3	4.0	4.0	0.33
no7_*	169.0	438.0	14.0	5/5	10.6	4.4	0.46
o[7-9]_*	183.2	466.9	14.7	9/9	8.7	3.2	0.66
pp-*	2960.0	2221.0	370.0	6/6	12.8	12.8	0.17
robust_*	253.9	298.3	2.0	420/420	6.8	6.8	0.67
sched_*	7361.5	3685.0	1.5	4/4	12.5	10.0	0.35
shortfall_*	249.9	294.3	2.0	420/420	7.8	7.8	0.37
slay*	393.0	936.0	14.0	14/14	8.6	7.0	0.27
sssd-*	286.0	314.5	18.0	16/16	7.6	5.6	0.42
stolpe07-*	1483.0	1202.0	164.7	3/3	29.0	18.3	0.71
tls*	521.4	989.8	51.6	5/6	7.6	7.0	0.49
turbine*	201.6	289.3	52.4	7/7	4.9	4.9	0.30
uflquad-*	8011.0	6811.0	1333.3	3/4	15.3	14.7	0.19
wiener_*	2297.5	2508.5	71.3	41/45	12.4	9.9	0.43
misc	1534.1	1374.5	158.9	11/14	10.5	7.6	0.33

Table 2.5. Results for CBLIB instances, averaged per problem group, $\epsilon_{\text{tol}} = 10^{-5}$. "Problem subset": name of problem group; "# var": number of variables; "# con": number of linear constraints; "# soc": number of second-order cone constraints; "solved/total": number of solved vs. total instances; "total iter": number of iterations in Algorithm 3; "SQP iter": number of iterations in which NLP-SQP step was accepted; "iter warm/iter cold": iterations for warm start divided by iterations for cold start (only for instances solved in both settings).

to failures of the QP subproblem solver, and Algorithm 3 exceeded the maximum number of 200 iterations in 2 cases. In comparison, MOSEK, with default settings, failed on 5 instances (those were solved correctly with Algorithm 3), incorrectly declared 3 instances to be

infeasible, and labeled 6 instances to be unbounded (of which 3 were solved by Algorithm 3). We observed that some instances, especially those in the `clay*`, `fo[7-9]*`, `m[3-9]*`, `no7*`, `o[7-9]_*` subsets, are degenerate, having an optimal objective function value of 0, and the assumption necessary to prove fast local convergence is violated. This matches the observations in the table, where the SQP step was accepted only in a relatively small fraction of the iterations.

To showcase the warm-starting feature of the algorithm, we took the 1,563 previously successfully solved instances, perturbed 10% of the entries of the final primal-dual iterate by a random perturbation, uniformly chosen in $[-0.1, 0.1]$, and used this as the starting point for a warm-started run. Here, QP subproblem failure occurred in 3 cases and 2 instances exceeded the iteration limit. As we can see from the last column in Table 2.5, on average, the number of iterations was reduced in most cases, often by more than 50%.

2.6. Concluding remarks

We presented an SQP algorithm for solving SOCPs and proved that it converges from any starting point and achieves local quadratic convergence for non-degenerate SOCPs. Our numerical experiments indicate that the algorithm is reliable, converges quickly when a good starting point is available, and produces more accurate solutions than a state-of-the-art interior-point solver.

Future research would investigate whether the proposed algorithm is a valuable alternative for interior-point methods for small problems or for the solution of a sequence of related SOCPs. An efficient implementation of the algorithm beyond our Matlab prototype would be tightly coupled with a tailored active-set QP solver that efficiently adds or removes cuts instead of solving each QP essentially from scratch. Parametric active-set solvers such as

qpOASES [**25**] or QORE [**70**] might be suitable options since they do not require primal or dual feasible starting points.

CHAPTER 3

RestartSQP: A Sequential Qudratic Programming Solver

3.1. Introduction

This chapter presents a new open-source software package called RestartSQP. This software package implements a Sequentially Quadratic Programming (SQP) method designed to solve nonlinear optimization problems (NLP) of the form given in (1.1), that is

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad c(x) = 0, \quad d(x) \leq 0.$$

In this formulation, the objective function $f : \mathbb{R}^n \to \mathbb{R}$, as well as the equality constraints $c : \mathbb{R}^n \to \mathbb{R}^{m_c}$ and inequality constraints $d : \mathbb{R}^n \to \mathbb{R}^{m_d}$, are assumed to be twice-continuously differentiable.

SQP methods and interior-point methods are two commonly used approaches for handling constrained problems of the form (1.1). SQP methods aim to find an approximate solution by iteratively solving a sequence of quadratic programming (QP) subproblems. On the other hand, interior-point methods seek to construct a continuous path that leads to an optimal solution of (1.1). This path is typically parameterized by a positive value called μ, which can be interpreted as a perturbation applied to the optimality conditions of the problem. When solving nonlinear programming problems, SQP methods typically focus on solving linear equations involving a subset of variables and constraints. In contrast, interior-point methods solve linear systems that incorporate all the constraints and variables.

Interior point methods are widely recognized for their effectiveness in solving large-scale nonlinear optimization problems. However, despite previous research on the topic (e.g.,[**36, 84, 37**]), the utilization of a "warm start" remains challenging within the framework of interior point methods in practice. The interior point methods pose challenges when it comes to adapting them to solve a sequence of related NLP problems. This difficulty can be attributed to the "path-following" interpretation of interior-point methods. In the neighborhood of an optimal solution, taking a step along the primal path $x(\mu)$ of the perturbed solution is well-defined. However, attempting to take a step onto this path from a neighboring point becomes highly sensitive to perturbations. This sensitivity can lead to numerical instability and pose challenges in maintaining the desired convergence properties when solving a sequence of related NLP problems.

Compared to the interior-point method, SQP methods have the ability to take advantage of a good initial starting point. In particular, when active-set QP solvers (discussed in Chapter 3.2.2) are employed, SQP methods can rapidly converge to the optimal solution with a few numbers of iterations, given a near-optimal point as an initial input. Consequently, SQP methods are highly suitable for addressing a sequence of related NLP problems, as the optimal solution obtained from the previous NLP can be effectively leveraged to efficiently solve subsequent NLPs. This can be particularly useful in many applications, including optimal control and trajectory planning [**62, 10**], hydropower reservoir modeling [**22**], air traffic management [**66**], multiperiod AC optimal power flows [**50, 56, 68**] and many more.

The software implements Fletcher's Sℓ_1QP method [**27**] and utilizes the parametric active-set method to solve QP subproblems. The parametric active-set method is specifically tailored to handle sequences of QP problems, making it highly suitable for this purpose. The versatility of RestartSQP extends beyond its role as a regular NLP solver, as it also offers the

distinctive capability of "warm-starting" NLPs. This means it can handle dynamic changes in constraint values and seamlessly incorporate additional constraints during the optimization process. Moreover, RestartSQP supports a crossover feature from interior point method solvers like Ipopt.

3.1.1. Overview

This chapter is structured as follows: In Chapter 3.2, we provide preliminary information on SQP methods and QP methods to establish a foundation for the subsequent discussions. Chapter 3.3 presents an overview of the Sℓ_1QP algorithm that has been implemented in the RestartSQP software and the updating rule for adjusting the penalty parameter. In Chapter 3.4, we delve into the implementation details of RestartSQP. Finally, in Chapter 3.5, we present the numerical results obtained from applying RestartSQP to various problems. This includes benchmarking against the CUTE NLP test set [**12**] and conducting warm-start experiments to assess the performance and effectiveness of the software.

3.2. Preliminaries

3.2.1. The SQP method

SQP methods have been one of the active research topics in the field of optimization and one of the most efficient methods for solving nonlinear constrained optimization problems of the form (1.1). Its origins can be traced back to the work by Wilson [**83**] in 1963. Subsequently, the method gained popularity through the works of Biggs [**8**], Han [**43**], and Powell [**64**], who extended its applicability to general nonlinear constraints during the 1970s. For a more comprehensive history of SQP methods and a detailed list of references, interested readers can refer to [**34, 27, 59, 18**].

Over the past few decades, a number of reliable and efficient general-purpose SQP solvers have been developed and have proven their effectiveness. These solvers have been widely used in various applications and have demonstrated their ability to efficiently solve nonlinear constrained optimization problems. Some notable examples include NLPQLP [**69**], SNOPT[**33**], KNITRO [**82**].

The basic idea of the SQP method is to model the NLP at a given approximate solution x^k by the quadratic program (QP) subproblem

$$\begin{aligned} &\min_{p\in\mathbb{R}^n} \nabla f(x^k)^T p + \tfrac{1}{2} p^T H^k p \\ &\text{s.t. } c(x^k) + \nabla c(x^k)^T p = 0, \\ &\qquad d(x^k) + \nabla d(x^k)^T p \leq 0, \end{aligned} \tag{3.1}$$

where H^k is a symmetric matrix. It then uses the solution p^k to construct a better approximate solution x^{k+1}. This process is iterated to create a sequence of iterates that will converge to x^*.

The local convergence of SQP methods relies on having knowledge of the active inequality constraints of the NLPs 1.1 at the local minimum x^*. This assumption can be supported by the following theorem.

Theorem 3.2.1 ([**67**])**.** *Given x^* a local solution of* (1.1) *at which the KKT conditions are satisfied for some vectors (λ^*, μ^*). Suppose that LICQ, strict complementarity, and second-order sufficient conditions hold at (x^*, λ^*, μ^*). If (x^k, λ^k, μ^k) is sufficiently close to (x^*, λ^*, μ^*), then there is a local solution of the QP subproblem* (3.1) *whose active set $\mathcal{A}^k$ is the same as the active set $\mathcal{A}(x^*)$ of the nonlinear program* (1.1)*.*

The SQP method possesses several important properties that are worth noting. First, it is important to recognize that the SQP method is not a feasible-point method. This means that the points in the iterate sequence $\{x^k\}$ generated by the SQP method do not necessarily need to be feasible with respect to the constraints of the nonlinear programming problem (NLP). In other words, the iterates do not have to satisfy the constraints during the course of the optimization process.

Second, by employing active-set QP solvers for the subproblems, the SQP method has the advantage of leveraging a good initial starting point. This capability enables the method to perform "warm-starting", wherein it utilizes optimal information obtained from solving a nearby NLP to accelerate the solution process for a subsequent NLP. By incorporating this warm-start strategy, the SQP method can significantly improve computational efficiency and convergence speed.

However, it is important to acknowledge that the SQP method is primarily designed for small to medium-scale problems. Each iteration of the SQP method involves solving a QP subproblem, which can be computationally expensive. Therefore, for large-scale problems, the SQP method may not be the most efficient approach. The computational cost associated with solving the QP subproblems should be taken into consideration when deciding on the appropriate optimization method.

Despite the potential computational cost, the SQP method often converges in fewer number of function/gradient evaluations compared to interior point methods. This property makes it particularly advantageous in situations where the evaluations of the objective function and its gradients dominate the computational cost of solving the optimization problem.

3.2.2. QP methods

The efficiency of SQP methods relies heavily on the availability of fast and accurate algorithms for solving QP subproblems. Methods for solving QPs can be broadly categorized into two main types: active-set methods and interior-point methods.

Active-set methods aim to identify the correct active set, which consists of the constraints that are satisfied with equality at the optimal solution of the QP. One advantage of active-set methods is their suitability for "warm-starting". This means that if a good estimate of the optimal active set or a favorable starting point is provided, the algorithm can converge to the optimal solution with only a few iterations.

However, it is worth noting that active-set methods also have their limitations. They may encounter challenges when dealing with degenerate constraints. Additionally, for large-scale problems with a significant number of constraints, the computational cost of active-set methods can become prohibitive. In contrast to active-set methods, interior-point methods offer an alternative approach for solving QPs. The interior-point QP solvers can be useful for handling large-scale QP instances. Therefore, the choice between these methods depends on the problem characteristics, problem size, and available computational resources. Over the past few decades, numerous solvers have been developed, demonstrating the importance and popularity of this approach. A detailed list of software for solving QPs can be found in [**39**].

One relatively recent approach to solving QP problems is the parametric active-set method. This method is particularly suitable for situations where prior knowledge or information can be utilized to expedite the QP solution or when high solution accuracy is required. It is well-suited for SQP methods, where a sequence of QP problems needs to be

solved. The fundamental concept behind this method is to trace the solution of a linear homotopy parameterized by $\tau \in [0,1]$, connecting an optimization problem with a known solution at $\tau = 0$ to the problem that needs to be solved at $\tau = 1$. This approach, initially proposed by Best [**7**] and subsequently implemented by Ferreau et al. [**25**] and Schork [**70**], aims to solve the following QP problems:

$$\begin{aligned} &\min_{x(\tau)\in\mathbb{R}^n} \tfrac{1}{2} x(\tau)^T H x(\tau) + g(\tau)^T x(\tau) \\ &\text{s.t. } Ax(\tau) \leq b(\tau). \qquad (\lambda(\tau)) \end{aligned} \tag{3.2}$$

Here, $A \in \mathbb{R}^{m\times n}$, $H \in \mathbb{R}^{n\times n}$, $g(\tau)$, and $b(\tau)$ are affine-linear functions of the parameter $\tau \in [0,1]$. It is assumed that $m > n$. Before delving into the algorithm's specifics, let us introduce the concept of the working set within the context of (3.2).

Definition 3.2.1 (Active set for (3.2))**.** *For a given $\tau \in [0,1]$ and $x(\tau) \in \mathbb{R}^n$, the index set*

$$\mathcal{A}(x(\tau)) := \{i \in [m] : a_i^T x(\tau) = b_i(\tau)\}$$

is referred to as the active-set $\mathcal{A}(\tau)$ at $x(\tau)$, where a_i is the i-th row vector of A.

Definition 3.2.2 (Working set for (3.2))**.** *For a given $\tau \in [0,1]$ and $x(\tau) \in \mathbb{R}^n$, the working set $\mathcal{W}(\tau)$ is defined as the subset of the active set $\mathcal{A}(x(\tau))$, where the vectors $\{a_i\}$ for $i \in \mathcal{W}(\tau)$ are linearly independent.*

The parametric QP method is an active set method that involves maintaining the working set $\mathcal{W}(\tau)$ and its corresponding Lagrangian multiplier vector $\lambda(\tau)$ for the active constraints. During the solution process, the method identifies a finite number of breakpoints $\tau^0, \cdots, \tau^K$ with $K > 0$ satisfying the following ordering:

$$0 = \tau^0 < \tau^1 < \cdots < \tau^{M-1} < \tau^K = 1.$$

The method initiates with $\tau^0 = 0$, utilizing a known primal-dual solution $(x(0), \lambda(0))$ and the corresponding working set $\mathcal{W}(0)$. At each iteration, denoted by k, the method determines search directions to update the primal variable $x(\tau^{k+1})$, its multiplier vector $\lambda(\tau^{k+1})$, and the working set $\mathcal{W}(\tau^{k+1})$. The working set is modified by adding, deleting, or exchanging constraints at the end of each interval. The method terminates when the optimal solution is achieved at $\tau = 1$, or further increments in τ lead to either an infeasible problem or an unbounded objective.

It is worth noting that the parametric QP method is capable of handling scenarios where the Hessian matrix H and constraint matrix A change from one problem to another. Suppose we have an optimal primal-dual solution (x^*, λ^*) and an optimal working set $\mathcal{W}^*$ for the QP problem:

$$\min_{x \in \mathbb{R}^n} \tfrac{1}{2} x^T H x + g^T x \quad \text{s.t} \quad Ax \leq b$$

and we aim to solve another QP problem:

$$\min_{x \in \mathbb{R}^n} \tfrac{1}{2} x^T \tilde{H} x + \tilde{g}^T x \quad \text{s.t} \quad \tilde{A}x \leq \tilde{b}$$

with new matrices $\tilde{H}$, $\tilde{A}$, vectors $\tilde{g}$, and $\tilde{b}$. In this case, we can define the affine linear functions g and b as follows: $b(0) = b + (\tilde{A} - A)x^*$, $b(1) = \tilde{b}$, $g(0) = g - (\tilde{H} - H)x^* + (\tilde{A} - A)^T \lambda^*$, and $g(1) = \tilde{g}$. Additionally, the initial working set can be set as $\mathcal{W}(0) = \mathcal{W}^*$.

An important property of the parametric QP method is that it requires initialization with the optimal solution of a QP. In other words, the solution and working set corresponding to $\tau = 0$ need to be provided. When a sequence of QP problems is solved, the solution and

working set from the preceding problem are commonly used to initialize the next problem. For the first QP in a sequence, an auxiliary problem with a trivial solution is typically defined. For instance, given H and A, one can define the QP with $b(0) \geq 0$ and $g(0) \geq 0$. Then $(x(0), \lambda(0)) = (0, 0)$ can be the optimal solution for this auxiliary QP.

Because of this property, the utilization of parametric QP methods offers a notable advantage for solving QPs. Unlike other methods that may require a feasible solution to start the algorithm, parametric QP methods do not necessarily need a primal or dual feasible solution for initialization. This eliminates the need for a Phase-1 procedure commonly used in primal or dual active-set methods.

However, it is important to note that the working set $\mathcal{W}(0)$ for this auxiliary problem may differ significantly from the optimal working set $\mathcal{W}(1)$ for the first QP we aim to solve. Consequently, the initialization phase, which involves solving the auxiliary QP before proceeding to the first QP, may require a considerable number of iterations. This initialization process is often referred to as a "cold start" since it begins with an unfavorable starting point. On the other hand, if we define QP(0) as a closed QP problem to QP(1) during the initialization phase, with a known optimal solution $(x(0), \lambda(0))$ and optimal working set $\mathcal{W}(0)$, it can significantly reduce the number of iterations and facilitate a "warm start" for the entire sequence of QPs. This warm-start property is particularly valuable when considering "warm-starting" NLPs, as discussed in detail in Chapter 3.4.5.

The parametric QP method, with its ability to exploit these warm-starting properties, is highly suitable for applications where warm-starting is desired. It naturally fits within SQP methods for solving sequences of problems.

3.3. Algorithm

In this chapter, we describe the Sℓ_1QP method introduced by Fletcher [**27**]. The algorithm, outlined in Algorithm 4, aims to convert a constrained optimization problem into an unconstrained problem using the exact ℓ_1 penalty function defined as:

$$\phi_1(x;\pi) = f(x) + \pi(\|c(x)\|_1 + \|[d(x)]^+\|_1),$$

where $\pi > 0$ is the penalty parameter and $[d(x)]^+$ denotes the positive part of $d(x)$.

Let us define:

$$\begin{aligned} q^k(x;\pi) =& f(x^k) + \nabla f(x^k)^T(x-x^k) + \tfrac{1}{2}(x-x^k)^T H^k(x-x^k) + \\ & \pi\big(\|c(x^k) + \nabla c(x^k)^T(x-x^k)\|_1 + \|[d(x^k) + \nabla d(x^k)^T(x-x^k)]^+\|_1\big) \end{aligned}$$

where H^k is the Hessian of the Lagrangian function at x^k, given by

$$H^k = \nabla^2_{xx}\mathcal{L}(x^k,\lambda^k,\mu) = \nabla^2_{xx} f(x^k) + \sum_{i=1}^{m_c} \lambda_i \nabla^2_{xx} \nabla c_i(x^k) + \sum_{i=1}^{m_d} \mu_i \nabla^2_{xx} d_i(x^k), \tag{3.3}$$

or an approximation of it. Here λ and μ are the Lagrange multipliers associated with the equality c and inequality constraints d, respectively. The function q^k can be seen as an ℓ_1-penalty function with a local affine approximation for c and d, as well as a quadratic approximation for the objective function f.

Fletcher made the observation that the minimization of the ℓ_1-penalty function can be achieved by solving a sequence of non-differentiable unconstrained subproblems. These subproblems are given in the following form:

$$\min_{x\in\mathbb{R}^n} q^k(x;\pi).$$

To address this optimization task, Fletcher proposed the utilization of a trust-region method. In particular, for a fixed value of π, the corresponding trust-region subproblem takes the following form:

$$\min_{p\in\mathbb{R}^n} q^k(x^k+p;\pi) \quad \text{subject to} \quad \|p\|_\infty \le \Delta^k,$$

where $\Delta^k > 0$ represents the trust-region radius at iteration k.

This subproblem is equivalent to the following smooth quadratic program:

$$\min_{p\in\mathbb{R}^n,u\in\mathbb{R}^m} \nabla f(x^k)^T p + \tfrac{1}{2}p^T H^k p + \pi(e^T s_c^+ + e^T s_c^- + e^T s_d^+) \tag{3.4a}$$

$$\text{s.t.} \quad c(x^k) + \nabla c(x^k)^T p + s_c^- - s_c^+ = 0, \qquad (\hat{\lambda}^k) \tag{3.4b}$$

$$d(x^k) + \nabla d(x^k)^T p - s_d^+ \le 0, \qquad (\hat{\mu}^k) \tag{3.4c}$$

$$-\Delta^k e \le p \le \Delta^k e, \quad s_c^+,\ s_c^-,\ s_d^+ \ge 0, \tag{3.4d}$$

where e denotes a vector of ones. In this formulation, the auxiliary variables s_c^+ and s_c^- can be interpreted as the positive and negative parts, respectively, for the affine function $c(x^k) + \nabla c(x^k)^T p$. Similarly, s_d^+ represents the positive part of $d(x^k) + \nabla d(x^k)^T p$.

One advantage of this formulation is that the QP subproblem (3.4) is always feasible even when the linearized constraints are inconsistent in (3.1). This characteristic allows for the effective utilization of warm-start strategies, leading to reduced solution times in practical implementations.

After solving the QP subproblem (3.4), the algorithm determines whether to accept the step p^k by evaluating the ratio:

$$\theta^k = \frac{\phi_1(x^k;\pi) - \phi_1(x^k + p^k;\pi)}{q^k(x^k;\pi) - q^k(x^k + p^k;\pi)}. \tag{3.5}$$

The numerator in the ratio represents the actual reduction in the merit function between the current iterate x^k and the trial iterate $x^k + p^k$, while the denominator is the predicted reduction in the merit function. The ratio θ^k serves as a measure of the accuracy of the second-order approximation of the nonlinear programming problem within the trust region, ensuring the global convergence of the trust-region algorithm. The trial iterate $x^k + p^k$ is accepted if $\theta^k \geq \eta_s$, where $\eta_s \in (0, 1)$ is a constant.

Due to the implicit minimization of the ℓ_1-penalty function, there is a possibility of encountering the Maratos effect [**55**]. The Maratos effect is a phenomenon that can arise when using algorithms based on merit functions for optimization problems with an implicit ℓ_1-penalty function. It refers to the situation where these algorithms fail to converge rapidly because they reject steps that would make progress toward a solution. This rejection is typically due to the merit function's evaluation, which incorporates both the objective function and penalty terms for constraint violations.

When the penalty parameter is very large, the penalty terms can dominate the merit function, causing an excessive focus on meeting constraints rather than optimizing the objective. Consequently, the algorithm can become overly conservative and reject steps that could potentially lead to improvements in the objective function.

To address the issue posed by the Maratos effect, a second-order correction step is introduced as a mitigation strategy. This correction step aims to reduce the constraint violation and enable algorithms to overcome the difficulties associated with the Maratos effect. The

second-order correction step is computed by solving the following QP subproblem:

$$
\begin{aligned}
&\min_{s\in\mathbb{R}^n} \nabla f(x^k)^T(p^k+s) + \tfrac{1}{2}(p^k+s)^T H^k(p^k+s) + \pi(e^T s_c^+ + e^T s_c^- + e^T s_d^+) \\
&\text{s.t. } c(x^k+p^k) + \nabla c(x^k)^T s + s_c^- - s_c^+ = 0, && (\tilde{\lambda}^k) \\
&\qquad d(x^k+p^k) + \nabla d(x^k)^T s - s_d^+ \le 0, && (\tilde{\mu}^k) \\
&\qquad -\Delta^k e \le p^k + s \le \Delta^k e, \quad s_c^+,\ s_c^-,\ s_d^+ \ge 0.
\end{aligned}
\tag{3.6}
$$

Note that the second-order correction is only performed when the ratio test fails, i.e., when $\theta^k < \eta_s$ with the previously calculated step p^k. In such cases, a second-order correction is applied to refine the solution. After solving the second-order correction QP problem (3.6), the trial iterate is updated as $x^{k+1} = x^k + p^k + s^k$, and the ratio (3.5) is updated accordingly. A ratio test is then performed again to determine whether the trial iterate is accepted or rejected. For the discussion of the convergence properties of the trust-region method with the second-order correction steps, see [**85**].

Upon accepting or rejecting the trial iterate, the trust-region radius Δ^k is updated. Specifically, if the ratio θ^k is greater than a predetermined constant $\eta_e \in (0,1)$ and the step p^k is active at the trust-region bounds, the trust-region radius is increased. Conversely, if the ratio θ^k is smaller than a constant η_c with $\eta_c < \eta_e$, the trust-region radius is decreased.

The process continues until a stopping criterion is met, such as reaching a maximum number of iterations, satisfying an optimality condition, or achieving a desired level of accuracy.

It can be proved that, under certain regularity conditions, the Sℓ_1QP method can achieve a local quadratic convergence rate. For a comprehensive analysis of the convergence properties of the Sℓ_1QP method, please refer to the works by Fletcher [**27**] and Burke et al. [**15**].

3.3.1. Penalty parameter update

We want to note that the efficiency of the Sℓ_1QP method largely relies on the appropriate choice of penalty parameter π. If the π is too small, the algorithm may generate iterates away from the optimal solution. On the other hand, if π is too large, the algorithm will converge slowly as the merit function focuses more on minimizing the constraint violations. Thus, the value of π must be chosen carefully. In our implementation, we adopt the technique of updating penalty parameters described in [**16**] and [**59**, Chapter 18].

The detailed procedure of the penalty update technique is described in Algorithm 5. In the implementation, we perform the penalty update procedure after calculating the SQP step p^k in line 4 in Algorithm 4.

To explain this technique, we need to define the piecewise linear model of the constraint violation at an iterate x^k as

$$m^k(p) = \|c(x^k) + \nabla c(x^k)^T p\|_1 + \|[d(x^k) + \nabla d(x^k)^T p]^+\|_1.$$

The procedure begins by first computing SQP steps p^k from (3.4) with the previous value of the penalty parameter π^k. If $m^k(p^k) = 0$, i.e., the constraints (3.4b) and (3.4c) are satisfied at p^k, then the value of π^k is kept. Otherwise, the solution to the following optimization problem

Algorithm 4 Sℓ_1QP

Require: Initial starting point $x^0 \in \mathbb{R}^n$; dual multipliers $\lambda^0 \in \mathbb{R}^{m_c}, \mu^0 \in \mathbb{R}^{m_d}$; trust-region radius $\Delta^0 > 0$; maximum trust-region radius $\Delta_{\max} \geq \Delta^0$; initial penalty parameter $\pi^0 > 0$; constant $\eta_c, \eta_e \in (0,1)$ with $\eta_c < \eta_e$, $\eta_s \in (0, \frac{1}{2})$, $\gamma_c \in (0,1)$, $\gamma_e > 1$, $\epsilon_1, \epsilon_2 \in (0,1)$.

$k \leftarrow 0$.
while x^k not optimal **do**
  Evaluate H^k from (3.3) with (x^k, λ^k, μ^k).
  Solve (3.4) for $(p^k, \hat{\lambda}^k, \hat{\mu}^k)$.
  Compute the ratio

$$\theta^k = \frac{\phi_1(x^k;\pi) - \phi_1(x^k + p^k;\pi)}{q^k(x^k;\pi) - q^k(x^k + p^k;\pi)}.$$

  if $\theta^k \geq \eta_s$ **then** ▷ Successful iteration
    $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}) \leftarrow (x^k + p^k, \hat{\lambda}^k, \hat{\mu}^k)$.
  else
    Solve (3.6) for $(p^k, \tilde{\lambda}^k, \tilde{\mu}^k)$.
    Recompute the ratio

$$\theta^k = \frac{\phi_1(x^k.\pi) - \phi_1(x^k + p^k + s^k;\pi)}{q^k(x^k;\pi) - q^k(x^k + p^k + s^k;\pi)}.$$

    if $\theta^k \geq \eta_s$ **then** ▷ Successful iteration
      $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}) \leftarrow (x^k + p^k + s^k, \tilde{\lambda}^k, \tilde{\mu}^k)$.
    else ▷ Unsuccessful iteration
      $(x^{k+1}, \lambda^{k+1}, \mu^{k+1}) \leftarrow (x^k, \lambda^k, \mu^k)$.
    end if
  end if
  if $\theta^k < \eta_c$ **then**
    $\Delta^{k+1} = \gamma_c \|p^k\|$. ▷ Decrease trust-region radius
  else
    if $\theta^k > \eta_e$ and $\|p\|_\infty = \Delta^k$ **then**
      $\Delta^{k+1} \leftarrow \min\{\gamma_e \Delta^k,\ \Delta_{\max}\}$. ▷ Increase trust-region radius
    else
      $\Delta^{k+1} \leftarrow \Delta^k$.
    end if
  end if
  $k \leftarrow k + 1$.
end while
$(x^*, \lambda^*, \mu^*) \leftarrow (x^k, \lambda^k, \mu^k)$
return Optimal primal-dual solution (x^*, λ^*, μ^*).

$$\min_{p \in \mathbb{R}^n} m^k(p) \qquad \text{s.t.} \quad \|p\|_\infty \leq \Delta^k, \tag{3.7}$$

will be required to determine the value of the new penalty parameter π^{k+1}. Let its optimal solution denote by p^∞. If $m^k(p^\infty) = 0$, it means that the linearized constraints are feasible within the trust region. The procedure then choose the value of $\pi^{k+1} > \pi^k$ such that $m^k(p(\pi^{k+1})) = 0$, where $p(\pi^{k+1})$ is the optimal solution of (3.4) with the penalty parameter value π^{k+1}. If $m^k(\infty) > 0$, then choose π^{k+1} that satisfies

$$m^k(0) - m^k(p(\pi^{k+1})) \geq \gamma_1(m^k(0) - m^k(p^\infty)). \tag{3.8}$$

for some $\gamma_1 \in (0, 1)$. That is, the reduction of m^k from the new SQP step $p(\pi^{k+1})$ is at least a fraction of the reduction given by p^∞.

In addition to this requirement, the updating strategy also needs the penalty function to significantly reduce the infeasibility measure. This reduction is important to encourage the acceptance of a good step. Sometimes, a step $p(\pi^{k+1})$ calculated by solving (3.4) with a particular π^{k+1} may satisfy $m(p^{k+1}) = 0$ or the condition given by (3.8), but only result in a small decrease in the quadratic model q^k. In such cases, the nonlinearity of the objective and constraints could overshadow this minor improvement. Consequently, the step $p(\pi^{k+1})$ might cause the penalty function ϕ_1 to increase, leading to its rejection. To prevent this undesirable behavior, the updating strategy further requires that after computing a trial value π^{k+1} and a search direction $p^k = p(\pi^{k+1})$, the following condition must hold:

$$q^k(x^k; \pi^{k+1}) - q^k(x^k + p^k; \pi^{k+1}) \geq \gamma_2 \pi^{k+1}(m^k(0) - m^k(p^k))$$

where $\gamma_2 \in (0,1)$ is a predetermined constant.

Algorithm 5 Penalty Update

Require: SQP step p^k from line 4 of Algorithm 4; previous penalty parameter π^k; $\gamma_1,\ \gamma_2 \in (0,1)$; initial parameter for penalty parameter increment $\beta > 1$.

1: **if** $m^k(p^k) = 0$ **then**
2: Set $\pi^{k+1} \leftarrow \pi^k$.
3: **else**
4: Compute p^∞ by solving (3.7).
5: $p(\pi^{k,0}) \leftarrow p^k$, $\pi^{k,0} \leftarrow \pi^k$.
6: $l \leftarrow 0$.
7: **if** $m^k(p^\infty) = 0$ **then**
8: **while** $m^k(p(\pi^{k,l})) > 0$ **do**
9: $\pi^{k,l+1} \leftarrow \beta \cdot \pi^{k,l}$.
10: Solve (3.4) with $\pi^{k,l+1}$ for $(p(\pi^{k,l+1}), \hat{\lambda}^k, \mu^k)$.
11: $l \leftarrow l + 1$.
12: **end while**
13: **else**
14: **while** $m^k(0) - m^k(p(\pi^{k,l})) < \gamma_1(m^k(0) - m^k(p^\infty))$ **do**
15: $\pi^{k,l+1} \leftarrow \beta \cdot \pi^{k,l}$.
16: Solve (3.4) with $\pi^{k,l+1}$ for $(p(\pi^{k,l+1}), \hat{\lambda}^k, \hat{\mu}^k)$.
17: $l \leftarrow l + 1$.
18: **end while**
19: **end if**
20: $p^k \leftarrow p(\pi^{k,l})$, $\pi^{k+1} \leftarrow \pi^{k,l}$;
21: **end if**
22: **while** $q^k(x^k; \pi^{k+1}) - q^k(x^k + p^k; \pi^{k+1}) < \gamma_2 \pi^{k+1}(m^k(0) - m^k(p^k))$ **do**
23: $\pi^{k+1} \leftarrow \beta \cdot \pi^{k+1}$.
24: Solve (3.4) with the new value of π^{k+1} for $(p^k, \hat{\lambda}^k, \hat{\mu}^k)$.
25: **end while**
26: **return** Updated SQP step p^k and dual multipliers $(\hat{\lambda}^k, \hat{\mu}^k)$; new penalty parameter π^{k+1}

3.4. Details of the Implementation

Before going into the details of the implementation, we want to note that the formulation described in (1.1) may be generalized in a straightforward manner to a form where x and c have both upper bound and lower bound. From now on, we will consider the generalized

case of the NLP problem in the following form:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{3.9a}$$

$$\text{s.t. } c_L \leq c(x) \leq c_U, \qquad (\lambda_c) \tag{3.9b}$$

$$x_L \leq x \leq x_U, \qquad (\lambda_x) \tag{3.9c}$$

where $c : \mathbb{R}^n \to \mathbb{R}^m$, $f : \mathbb{R}^n \to \mathbb{R}$ are twice continuously differentiable, $c_L \leq c_U$ and $x_L \leq x_U$. Note that the components of c_L and x_L can be $-\infty$ and components of c_U and x_U can be $+\infty$.

In this chapter, we present a comprehensive account of the implementation of the basic algorithm outlined in Algorithm 4. We will cover key aspects such as iterate initialization, the ratio test, termination checks, and the efficient handling of "warm-starting" NLPs in the software. We also discuss the lazy cut generation approach, which involves generating constraints on demand during the optimization process. This technique can significantly reduce the computational burden by selectively introducing constraints only when necessary.

3.4.1. Shifting the initial starting points

In our implementation, we shift the initial starting points x^0 if necessary such that the bound constraints are always satisfied. Specifically, let $\tilde{x}^0$ be the given starting point at the beginning of the algorithm, we set

$$x_i^0 = \begin{cases} x_{L,i}, & \text{if } \tilde{x}_i^0 < x_{L,i}, \\ x_{U,i}, & \text{if } x_{U,i} < \tilde{x}_i^0, \\ \tilde{x}_i^0 & \text{otherwise,} \end{cases}$$

for $i = 1, \cdots, n$. This shifting is done to ensure that the initial starting point is feasible for the bound constraints. The shifted starting point x^0 is then used to initialize the algorithm.

By that, we can reduce the number of constraints of the QP subproblem applied to (3.9) by combining the trust-region constraint with the linearized constraint applied to (3.9c). The QP subproblem for iteration k is then given by

$$
\begin{aligned}
\min_{p\in\mathbb{R}^n, u,v\in\mathbb{R}^m} \quad & \nabla f(x^k)^T d + \tfrac{1}{2} p^T H^k p + \rho e^T (u+v) \\
\text{s.t.} \quad & c_L \le c(x^k) + \nabla c(x^k)^T p + u - v \le c_U, \\
& \max(-\Delta^k,\ x^k_{L,i} - x^k_i) \le p_i \le \min(\Delta^k,\ x^k_{U,i} - x^k_i), \quad i \in [n], \\
& u, v \ge 0.
\end{aligned}
\tag{3.10}
$$

where u and v are the slack variables for the linearized constraints applied to (3.9b). With this formulation, an SQP search direction p^k can be computed in the way that every iterate x^k satisfies bound constraints. In addition, the linearized constraints violation for a search direction p will be computed as

$$
m^k(p) = \|[c_L - c(x^k) - \nabla c(x^k)^T p]^+\|_1 + \|[c(x^k) + \nabla c(x^k)^T p - c_U]^+\|_1.
$$

3.4.2. Ratio test

To handle round-off errors, we replace the criterion in the step 11 of Algorithm 4 by

$$
\Big(\phi_1(x^k;\pi) - \phi_1(x^k + p^k;\pi) + \epsilon^k\Big) \ge \eta_s \cdot \Big(q^k(x^k;\pi) - q^k(x^k + p^k;\pi) + \epsilon^k\Big),
$$

where

$$
\epsilon^k = 10^{-10} \cdot \max\{|f(x^k)|, m^k(0)\}.
$$

In situations where the predicted reduction is non-positive, i.e., $q^k(x^k;\pi) - q^k(x^k+p^k;\pi) \leq 0$, the algorithm takes the following actions: it forbids the step p^k, throws exceptions, and terminates the algorithm.

3.4.3. Termination checks

To determine an iterate x^k is optimal, we perform a check to verify if the primal-dual pair $(x^k, \lambda_c^k, \lambda_x^k)$ satisfies the KKT conditions.

Specifically, we compute violations of primal feasibility, stationarity, and complementarity to determine optimality. This step is performed immediately after the acceptance of a new step. It is worth noting that since the iterate x^k is computed from the QP subproblem (3.10), there is no need to check the feasibility of the bound constraints (3.9c) or the correctness of the signs of the corresponding multipliers. The violations of primal feasibility and complementarity are computed as follows:

$$
\begin{aligned}
v_{\texttt{primal}} &= \max\left\{\|[c_L - c(x^k)]^+\|_\infty, \|[c(x^k) - c_U]^+\|_\infty\right\}, \\
v_{\texttt{compl}} &= \max_{i\in[n], j\in[m]} \begin{Bmatrix} \min\left\{[\lambda_{x,i}^k]^+,\ x_i^k - x_{L,i}\right\}, \min\left\{[-\lambda_{x,i}^k]^+,\ x_{U,i} - x_i^k\right\} \\ \min\left\{[\lambda_{c,j}^k]^+,\ c_j(x^k) - c_{L,j}\right\}, \min\left\{[-\lambda_{c,j}^k]^+,\ c_{U,j} - c_j(x^k)\right\} \end{Bmatrix}.
\end{aligned}
$$

For checking the violation for the dual feasibility, we calculate it by

$$
v_{\texttt{stat}} = \frac{\left\|\nabla f(x^k) + \nabla c_j(x^k)\lambda_c^k - \lambda_x^k\right\|_\infty}{\max\left(1, \frac{\|\lambda_x^k\|_1 + \|\lambda_c^k\|_1}{100(n+m)}\right)}.
$$

The algorithm determines that the point x^k satisfies the KKT conditions if the maximum violation among the three quantities is below a user-defined tolerance $\tau_{\texttt{SQP}}$, i.e.,

$$\max\{v_{\texttt{stat}}, v_{\texttt{primal}}, v_{\texttt{compl}}\} \leq \tau_{\texttt{SQP}},$$

By default, the value of $\tau_{\texttt{SQP}}$ is set to 10^{-6}.

3.4.4. QP solvers

The software has been integrated with two parametric QP solvers, QPoases[**25**] and QORE[**70**]. In addition to the parametric QP method, we have also integrated the software with Ipopt[**81**], a state-of-the-art interior point method for nonlinear programming.

3.4.5. "Warm starts" of nonlinear programming

In this chapter, we describe the warm-start strategy employed in the software, focusing on the application of parametric quadratic programming (QP) methods discussed in Chapter 3.2.2. Before discussing the specific implementation techniques, it is crucial to establish a clear understanding of the concept of "warm-start" NLPs.

The warm-start strategy in our context involves utilizing information obtained from the optimal solution of a previous NLP as the initial starting point for the current optimization problem. By reusing this initial solution effectively, we can expedite the convergence process and improve overall computational efficiency.

Specifically, let us consider the following two NLPs:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f^0(x) \\ \text{s.t. } & c_L^0 \leq c^0(x) \leq c_U^0, \quad (\lambda_c^0) \\ & x_L^0 \leq x \leq x_U^0, \qquad (\lambda_x^0) \end{aligned} \tag{3.11}$$

and

$$
\begin{aligned}
&\min_{x\in\mathbb{R}^n} \quad f^1(x) \\
&\text{s.t. } c_L^1 \le c^1(x) \le c_U^1, \quad (\lambda_c^1) \\
&\qquad x_L^1 \le x \le x_U^1, \qquad (\lambda_x^1)
\end{aligned} \tag{3.12}
$$

where $f^1 : \mathbb{R}^n \to \mathbb{R}$, $f^1 : \mathbb{R}^n \to \mathbb{R}$, $c^0 : \mathbb{R}^n \to \mathbb{R}^m$ and $c^1 : \mathbb{R}^n \to \mathbb{R}^m$. We assume that $c_L^i \le c_U^i$ and $x_L^i \le x_U^i$ and f^i, c^i are twice continuously differentiable for $i = 0, 1$. We assume the optimal primal-dual solution $(x^{0,*}, \lambda_c^{0,*}, \lambda_x^{0,*})$ and the optimal working set $\mathcal{W}^{0,*}$ for the first NLP (3.11) are known.

In SQP methods, warm-starting involves taking the optimal solution $(x^{0,*}, \lambda_c^{0,*}, \lambda_x^{0,*})$ and optimal working set $\mathcal{W}^{0,*}$ obtained from the previous NLP problem, (3.11), as the initial starting point for the new NLP problem, (3.12). This previous solution information is used to set up the QP subproblems accordingly during the SQP iterations. By utilizing the previous NLP solution as a warm start, the SQP method can potentially converge faster and require fewer iterations to reach the optimal solution for the new NLP problem.

The parametric QP method offers numerous advantages when solving a sequence of QP problems, particularly during the solution process for QP subproblems within an SQP framework. It allows for efficient updates of the solution and working set from the previous QP problem, thereby facilitating a "warm start" for subsequent QPs in the sequence. In this context, if the QP solver's internal data for the previous NLP problem (3.11) is retained, the first SQP problem for the new NLP problem (3.12) can be considered as the subsequent QP in this sequence.

In scenarios where the QP solver's internal data for the previous NLP problem (3.11) is not available, such as when the previous NLP is not solved using the parametric QP method,

initializing the QP problem sequences for the new NLP problem (3.12) becomes necessary. The common approach is to use an auxiliary QP with a trivial solution, known as a "cold start" initialization as mentioned in Chapter 3.2.2. However, this cold start initialization may require a significant number of QP iterations before reaching the optimal solution for the first QP in the sequence. Despite the benefits of warm-starting provided by the parametric QP method, this initial phase of solving the auxiliary QP can still involve a substantial number of iterations.

To address the challenges posed by the cold start initialization and reduce the number of iterations required, we propose explicitly defining a QP problem using the optimal solution $(x^{0,}, \lambda_c^{0,}, \lambda_x^{0,*})$ and the optimal working set $\mathcal{W}^{0,*}$ obtained from the previous NLP solution. In this approach, we aim to set $\mathcal{W}^{0,*}$ as the optimal working set for the defined QP problem, allowing the parametric QP methods to leverage the known information from the previous NLP solution more effectively.

The QP problem is constructed as follows:

$$
\begin{aligned}
\min_{p\in\mathbb{R}^n, u,v\in\mathbb{R}^m} \quad & \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}^0(x^{0,*}, \lambda_c^{0,*}, \lambda_x^{0,*})p + \nabla f(x^{0,*})^T p + \rho e^T(u+v) \\
\text{s.t.} \quad & c_L^0 - c(x^{0,*}) \le \nabla c(x^{0,*})^T p + u - v \le c_U^0 - c(x^{0,*}) \\
& \max(-\Delta, x_{L,i}^0 - x_i^{0,*}) \le p_i \le \min(\Delta, x_{U,i}^0 - x_i^{0,*}), \quad i \in [n] \\
& u, v \ge 0.
\end{aligned}
$$

In this formulation, $\mathcal{L}^0$ represents the Lagrangian function for the previous NLP problem (3.11). The parameter $\rho > 0$ is chosen such that $\rho > \|\lambda_c^{0,*}\|_\infty$, and $\Delta > 0$ is the user-defined initial trust-region radius for the new NLP problem (3.12). The optimal solution for this QP problem is $(p, u, v) = (0, 0, 0)$ with the optimal working set $\mathcal{W}^{0,*}$. By utilizing this manually

defined QP problem, we can eliminate the cold start initialization and solve the first QP in the sequence more efficiently.

By employing the warm-start strategy at both the SQP solver level and QP solver level, the software can efficiently solve a sequence of NLPs. It is important to note that the effectiveness of the warm-start approach relies on the similarity between the solutions and optimal working sets of each NLP in the sequence. When the solutions and optimal working sets are close to each other, the warm-start strategy can significantly improve convergence speed and overall computational efficiency.

3.4.5.1. Cross-over from Ipopt. Interior-point methods are renowned for their effectiveness in solving large-scale optimization problems. However, one limitation of these methods is the lack of precise information regarding the active constraints at the solution. This can lead to less accurate solutions and sensitivity information. To address this, the software incorporates a crossover feature that transitions from the interior-point method to the SQP method at the estimated interior-point solution.

The crossover feature serves two purposes. First, it allows users to initially solve the NLP using the interior-point method, obtaining an estimated solution. Then, by transitioning to the SQP method, a more accurate solution can be obtained, along with the correct identification of the active set and sensitivity information. This improves the understanding of the problem's constraints and enhances the overall solution quality.

Moreover, the crossover feature is particularly valuable when solving a sequence of similar NLPs. The interior-point method can be employed for solving the initial NLPs in the sequence, providing efficient solutions to these problems. Subsequently, the crossover function is activated, enabling the SQP method to solve the subsequent NLPs.

To perform the crossover functionality, we first use Ipopt, an interior-point solver, to obtain the optimal solution x^*. Using this solution, the software explicitly identifies an estimate of the active set. The identified active set, along with the optimal solution, is then passed as initial input to the SQP method. Specifically, we employ the following formulas to determine an estimation of the active set for bound constraints and general constraints, denoted by $\mathcal{A}_x^*$ and $\mathcal{A}_c^*$ respectively:

$$\mathcal{A}_{x,i}^* = \begin{cases} \text{inactive}, & \frac{x_i^* - x_{L,i}}{\max(\tau_p, \lambda_{x,i}^*)} > \tau_{active} \text{ and } \frac{x_{U,i} - x_i^*}{\max(\tau_p, -\lambda_{x,i}^*)} > \tau_{active}, \\ \text{active above}, & x_{U,i} = x_{L,i} \text{ or } \tau_{active} \geq \frac{x_i^* - x_{l,i}}{\max(\tau_p, \lambda_{x,i}^*)} > \frac{x_{u,i} - x_i^*}{\max(\tau_p, -\lambda_{x,i}^*)}, \\ \text{active below}, & \text{otherwise}, \end{cases}$$

and

$$\mathcal{A}_{c,j}^* = \begin{cases} \text{inactive}, & \frac{c_j(x^*) - c_{L,j}}{\max(\tau_p, \lambda_{c,j}^*)} > \tau_{active} \text{ and } \frac{c_{U,j} - c_j(x^*)}{\max(\tau_p, -\lambda_{c,j}^*)} > \tau_{active}, \\ \text{active above}, & c_{U,j} = c_{L,j} \text{ and } \lambda_{c,j}^* \leq 0, \text{ or } \tau_{active} \geq \frac{c_j(x^*) - c_{L,j}}{\max(\tau_p, \lambda_j^*)} > \frac{c_{U,j} - c_j(x^*)}{\max(\tau_p, -\lambda_j^*)}, \\ \text{active below}, & \text{otherwise}. \end{cases}$$

By default, τ_{active} is set to be 10^{-4} and τ_p is set to be 10^{-16}. We choose to use the interior point method solver initially because it excels in cold starting for large-scale NLPs. In contrast, the SQP method with an active-set QP solver may require numerous QP iterations, making it less efficient compared to the interior point method.

It should be noted that the effectiveness of this warm-start strategy depends on the strict complementarity condition being satisfied at the optimal solution. However, in practical applications, we have observed that this approach encounters challenges when strict complementarity fails, such as the nonlinear programming problems from AC optimal power flow.

Moreover, it is important to consider that the cost associated with performing crossover depends on the specific problem at hand. In small or medium-scale problems, the additional computational cost of the crossover procedure is generally a small fraction of the total solution cost. In these cases, it is advantageous to employ crossover to achieve a more accurate solution. However, for large-scale or highly degenerate problems, the cost of crossover may become significant. Thus, it is recommended to carefully evaluate the trade-off between improved solution accuracy and the additional computational cost incurred by the crossover procedure in such scenarios.

3.4.6. Lazy constraints generations for nonlinear programming

In the software, we implemented a lazy constraint generation scheme for solving NLPs. This approach is particularly useful in two scenarios: when solving a sequence of NLPs with an increasing number of constraints, or when dealing with an NLP that has a large number of constraints. Examples include the SDP relaxation of ACOPF, which involves an exponential number of nonlinear constraints in the problem formulation [**46**].

The process, described in Algorithm 6, involves solving the NLP with a selected subset of constraints and successively adding additional constraints (referred to as "cuts") to the problem and resolving it. In particular, we solve

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t. } & c_{L,i} \leq c_i(x) \leq c_{U,i}, \quad i \in \mathcal{I} \subseteq [m], \\ & x_L \leq x \leq x_U, \end{aligned}$$

for a selected subset of constraints $\mathcal{I}$, and then add additional constraints to the problem and resolve it.

Algorithm 6 Lazy Constraint Generation for NLP

1: Choose an initial subset of constraints $\mathcal{I}^0 \subseteq [m]$ for the NLP, ensuring that the NLP remains bounded below.
2: Set $k \leftarrow 0$.
3: Solve the NLP with the constraints defined by $\mathcal{I}^0$, obtaining the solution $(x^k, \lambda_x^k, \lambda_c^k)$.
4: **while** $\mathcal{V}^k = \{i \in [m] \setminus \mathcal{I}^k : x^k \text{ violate constraint } c_i\}$ is not empty **do**
5: Choose a set of constraints indices $\hat{\mathcal{V}}^k \subseteq \hat{\mathcal{V}}^k$.
6: Update the constraint set: $\mathcal{I}^{k+1} \leftarrow \mathcal{I}^k \cup \hat{\mathcal{V}}^k$.
7: Solve the NLP with the constraints defined by $\mathcal{I}^{k+1}$, obtaining the solution $(x^{k+1}, \lambda_c^{k+1}, \lambda_x^{k+1})$.
8: Increment k: $k \leftarrow k + 1$.
9: **end while**
10: **return** Optimal solution: $x^* = x^k$.

The algorithm begins by selecting an initial subset of constraints, $\mathcal{I}^0$, which chosen in a way that the NLP remains bounded below and easy to solve. Then, in each iteration, the NLP is solved using the current set of constraints, and if any violated constraints are found, a subset of those constraints is added to the constraint set for the next iteration. This process continues until no more violated constraints are found. Finally, the optimal solution, denoted as x^*, is obtained as the solution from the last iteration.

When applying the lazy constraint generation scheme to a sequence of NLPs, it becomes essential to handle the "warm-start" transition from one NLP to the next. Specifically, between each pair of NLPs, the methods needs to transition from the NLP defined by (3.9), that is,

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t. } & c_L \leq c(x) \leq c_U, \quad (\lambda_c) \\ & x_L \leq x \leq x_U, \qquad (\lambda_x) \end{aligned}$$

to a new NLP with additional constraints defined by

$$\begin{aligned}
\min_{x\in\mathbb{R}^n} \quad & f(x) \\
\text{s.t. } & c_L \le c(x) \le c_U, \quad (\lambda_c) \\
& \tilde{c}_L \le \tilde{c}(x) \le \tilde{c}_U, \quad (\lambda_{\tilde{c}}) \\
& x_L \le x \le x_U, \qquad (\lambda_x)
\end{aligned}$$

where $\tilde{c} : \mathbb{R}^n \to \mathbb{R}^{\tilde{m}}$ are twice-continuously differentiable, $\tilde{c}_L$ and $\tilde{c}_U$ satisfies $\tilde{c}_L \le \tilde{c}_U$.

To achieve efficient warm-starting and avoid the cold-start initialization, a similar strategy as discussed in Chapter 3.4.5 is employed. Specifically, when the optimal solution $(x^*, \lambda_c^*, \lambda_x^*)$ and its optimal working set $\mathcal{W}^*$ for (3.9) are available, we construct an additional QP problem to handle the additional constraints and facilitate warm-starting when using the parametric QP solver. The QP problem is formulated as follows:

(3.14a)
$$\min_{p\in\mathbb{R}^n, u,v\in\mathbb{R}^m} \ \tfrac{1}{2} p^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda_c^*, \lambda_x^*) p + \nabla f(x^*)^T p + \rho e^T (u + v)$$

(3.14b)
$$\text{s.t.} \qquad c_L - c(x^*) \le \nabla c(x^*)^T p + u - v \le c_U - c(x^*)$$

(3.14c)
$$\min(\tilde{c}_{L,i} - \tilde{c}_i(x^*), -\epsilon) \le \nabla \tilde{c}_i(x^*)^T p + u_i - v_i \le \max(\tilde{c}_{U,i} - \tilde{c}_i(x^*), \epsilon), \quad i \in [\tilde{m}]$$

(3.14d)
$$\max(-\Delta, x_{L,i} - x_i^*) \le p_i \le \min(\Delta, x_{U,i} - x_i^*), \quad i \in [n]$$

(3.14e)
$$u, v \ge 0.$$

In this formulation, the constant $\epsilon > 0$ is carefully chosen such that the constraint (3.14c) is not active, ensuring that the optimal working set $\mathcal{W}^*$, along with the trivial solution $(p, u, v) = (0, 0, 0)$, remains optimal for (3.14). By employing this additional QP, the parametric QP solver can avoid cold-start initialization for the QP subproblem sequence in the

SQP algorithm. Instead, it utilizes the optimal information obtained from the previously solved NLPs.

This lazy constraint generation approach has been integrated with the cross-over strategy discussed in Chapter 3.4.5.1. Initially, the interior point solver Ipopt is invoked to solve the initial NLP defined by $\mathcal{I}^0$ in line 3. Following the completion of the first NLP solve, the algorithm transitions to the SQP method to handle additional constraints and proceeds to solve all subsequent NLPs using this approach.

The lazy constraint generation approach provides a more efficient solution for NLPs with a large number of constraints by avoiding the inclusion of all constraints initially. Instead, it incrementally incorporates necessary constraints based on the current solution, thereby reducing the computational burden. It is worth noting that the selection of the initial subset of constraints, $\mathcal{I}^0$, as well as the order in which constraints are added, can significantly impact the efficiency and quality of the solution.

This approach is particularly advantageous when dealing with NLPs that have a large number of constraints, especially when the number of constraints is exponentially large. In such cases, the gradual addition of constraints based on the current solution proves superior to solving the problem with all constraints using an NLP solver. However, it is important to consider that for NLPs with a small or medium-sized number of constraints, employing an NLP solver to solve the problem with all constraints may be more efficient and faster. Thus, the choice of approach should be made based on the specific problem characteristics and computational resources available.

Option	Description	Default
trust_region_ratio_decrease_tol	If ratio $\leq$ trust_region_ratio_decrease_tol, then the trust-region radius for the next iteration is decreased for the next iteration (η_c in Algorithm 4) .	1e-8
trust_region_ratio_accept_tol	If ratio $\geq$ trust_region_ratio_increase_tol and the search direction hits the trust-region boundary, the trust-region radius will be increased for the next iteration (η_e in Algorithm 4).	1e-8
trust_region_decrease_factor	Factor used to reduce the trust-region size (η_c in Algorithm 4).	0.5
trust_region_increase_factor	Factor used to increase the trust-region size (η_e in Algorithm 4).	2.0
trust_region_init_size	Initial trust-region radius (Δ^0 in Algorithm 4).	10.0
trust_region_max_value	Maximum value of trust-region radius allowed for the radius update ($\Delta_{\max}$ in Algorithm 4).	1e10
trust_region_min_value	Minimum value of trust-region radius allowed for the radius update.	1e-16
penalty_parameter_init_value	Initial value of the penalty parameter (π^0 in Algorithm 4).	10.0
penalty_update_tol	Tolerance for trigger penalty parameter update.	1e-6
penalty_parameter_increase_factor	Factor by which penalty parameter is increased (β in Algorithm 5).	10.0
gamma_1	Parameter for penalty update (γ_1 Algorithm 5).	0.1
gamma_2	Parameter for penalty update (γ_2 Algorithm 5).	1e-6
penalty_parameter_max_value	Maximum value of the penalty parameter.	1e8
penalty_iter_max	Maximum number of penalty parameter update allowed in a single iteration in the main algorithm.	200
penalty_iter_max_total	Maximum number of penalty paremter update allowed in total	100
perform_second_order_correction	Perform the second-order correction during the main iteration.	true
qp_solver_max_num_iterations	The maximum number of iterations for the QP solver in solving each QP.	1e5
lp_solver_max_num_iterations	The maximum number of iterations for the QP solver in solving each LP.	1e5

Table 3.1. Summary of algorithmic parameters in RestartSQP with their respective default values

3.5. Numerical Results

In this chapter, we present the numerical results. We tested the software on the benchmark NLP test set CUTE [**12**]. The numerical experiment is performed on a macOS Ventura Version 13.1 with a 2.3GHz Intel i9 processor and 16GB RAM.

RestartSQP is run on the test set using the default option values specified in Table 3.1. In additional to the termination criterion mentioned in the previous chapter, the algorithm will also stop if the trust-region becomes too small or if the penalty parameter becomes too small. The algorithm is executed with a CPU time limit of 3 minutes and an iteration limit of 3000. The test instances were solved using the AMPL interface, utilizing the `.nl`

files generated from the AMPL model provided by [**79**]. Specifically, we selected instances with a number of variables and constraints smaller than or equal to 50000, resulting in a total of 737 available instances for our analysis. However, for instances `argauss`, `grouping`, and `lewispol`, they were excluded from the analysis due to their limited degrees of freedom. Additionally, the instances `dallasl`, `dallasm`, `hs068`, and `hs069` were excluded due to evaluation errors encountered within the AMPL interface. `fletcbv3, fletchbv, indef, static3` are removed because they appeared unbounded below.

In the following chapters, we conducted a practical performance evaluation of RestartSQP using different QP subsolvers. Specifically, we first analyze the performance of RestartSQP with the QORE parametric QP solver and assess its warm-start capability. Next, we will examine the performance of the SQP solver utilizing the Ipopt interior point method solver as its QP subsolver. The evaluation is performed on the remaining 726 instances from the CUTE test set. For the evaluation, we utilized the starting points provided in the AMPL model whenever available. If not available, we use trivial starting points instead.

3.5.1. Using QORE as QP subsolver

We executed RestartSQP on the CUTE instances, employing QORE [**70**] as the QP subsolver. QORE was compiled with multiple linear solvers, including UMFPACK, MA27, MA57, and MA86. The configuration of QORE includes the following settings:

- The `qore_umpack_ordering` option was set to `cholmod`, specifying the ordering method used by the UMFPACK solver within QORE.
- The `qore_ma57_ordering` option was set to `amd`, determining the ordering method employed by the MA57 solver within QORE.

- We assigned a value of 1e-14 to the `qore_basis_repair_tol` option, controlling the tolerance used in the basis repair process of QORE.
- The threshold pivoting tolerance for both UMFPACK and MA57 solvers was set to 0.1, determining when a pivot is considered too small during the factorization process.
- The near-zero element drop tolerance for both UMFPACK and MA57 solvers was set to $2.22e-16$, specifying when a numerical value is considered close to zero during factorization.
- Lastly, the threshold pivoting tolerance for the MA27 solver was also set to 0.1, governing the selection of pivotal elements during the factorization process.

Among the 726 instances, RestartSQP successfully solved 689 of them within the designated time limit. Out of these, 572 instances were solved to optimal. The detailed statistics can be found in Table B.1. It is worth noting that most of the failures were attributed to the exception where the predicted reduction was non-positive. Additionally, there were 17 instances that encountered unclassified internal errors.

Here is a breakdown of the failures:

- 7 instances failed due to exceeding the maximum number of iterates. [1].
- 55 instances failed because the predicted reduction is non-positive. [2].
- 3 instances failed because the trust region radius became too small. [3].

[1]These instances are `cresc50`, `fletcher`, `himmelbd`, `hs063`, `minc44`, `osbornea`, and `polak2`
[2]These instances are `beale`, `broydn7d`, `bt4`, `cresc100`, `cresc132`, `cube`, `deconvu`, `dixmaang`, `dixmaanj`, `dixmaank`, `errinros`, `flosp2hh`, `flosp2hm`, `flosp2tm`, `genhumps`, `hadamals`, `hairy`, `hart6`, `heart6ls`, `heart8ls`, `hs024`, `hs029`, `hs036`, `hs037`, `hs038`, `hs056`, `humps`, `launch`, `loghairy`, `logros`, `methanl8`, `nonmsqrt`, `optmass`, `palmer1e`, `palmer3b`, `palmer3e`, `palmer4a`, `palmer4e`, `palmer5a`, `palmer5b`, `palmer5e`, `palmer6e`, `palmer7a`, `palmer7e`, `palmer8a`, `palmer8e`, `pfit1`, `pfit1ls`, `pfit2`, `pfit2ls`, `sinquad`, `spanhyd`, `svanberg`, `watson`, and `woods`
[3]These instances are `cantilvr`, `chemrctb`, and `himmelbj`

- 14 instances encountered failure because the penalty parameter became too large. [4].
- 14 instances failed because the QP solver exceeded the maximum number of iterations. [5].
- 17 instances terminated because of internal QP solver errors. [6].
- 4 instances failed due to QORE declaring the QP as unbounded. [7].
- 3 instances failed due to QORE declaring them infeasible. [8].

Currently, it is important to acknowledge that parametric QP solvers, like QORE, may not yet match the robustness and maturity of well-established interior-point QP solvers such as Ipopt. As QORE is still undergoing development, it lacks some of the mature error-handling capabilities found in established solvers like Ipopt. As RestartSQP keeps improving, we will continue working on enhancing and fine-tuning the parametric active-set method used in the SQP solver. We expect that these efforts will lead to a more reliable and effective performance of QORE as the subsolver for our SQP method in the future.

3.5.1.1. Warm start for perturbed programs. In this chapter, we present a result of warm-starting. The process begins by utilizing Ipopt with default options to generate solution $(x^*, \lambda_c^*, \lambda_{x_L}^*, \lambda_{x_U}^*)$ for the original problem. Here, $\lambda_{x_L}^*$ represents the optimal multiplier for the lower bound constraints, while $\lambda_{x_U}^*$ represents the multiplier for the upper bound constraints. The time limit for Ipopt is set to 3 minutes. Out of 724 instances, 697 are

[4]These instances are `artif`, `core1`, `core2`, `disc2`, `discs`, `hs013`, `huestis`, `optctrl3`, `optctrl6`, `s365mod`, `semicon1`, `semicon2`, `vanderm2`, and `vanderm4`
[5]These instances are `bloweya`, `bloweyc`, `britgas`, `clnlbeam`, `cvxqp3`, `gausselm`, `haifam`, `hvycrash`, `noncvxu2`, `nuffield2`, `orthrds2`, `reading1`, `ubh1`, and `vanderm1`
[6]These instances are `aug2dcqp`, `bloweyb`, `catenary`, `dittert`, `drcavty3`, `eigenbco`, `haldmads`, `hanging`, `hs108`, `lch`, `minperm`, `nuffield`, `porous1`, `qpcstair`, `sawpath`, `twirism1`, and `vanderm3`
[7]These instances are `aug2dqp`, `hs084`, `lakes`, and `lminsurf`
[8]These instances are `dixmaand`, `kissing`, and `scosine`

solved to either optimality or acceptable levels within the time limit. The optimal multiplier λ_x^* is obtained by calculating $\lambda_{x_L}^* - \lambda_{x_U}^* *$.

To introduce variations, we perturb the optimal primal-dual solution obtained from Ipopt. Random uniform perturbations are generated for 10% of the entries in each vector $(x^*, \lambda_c^*, \lambda_{x_L}^*, \lambda_{x_U}^*)$, with perturbation values ranging from -10^{-3} to 10^{-3}. The resulting perturbed optimal solution is denoted as $(x^{perturbed}, \lambda_c^{perturbed}, \lambda_{x_L}^{perturbed}, \lambda_{x_U}^{perturbed})$. Furthermore, we calculate the perturbed multiplier $\lambda_x^{perturbed}$ by subtracting $\lambda_{x_U}^{perturbed}$ from $\lambda_{x_L}^{perturbed}$. These perturbed solutions serve as the initial primal-dual starting point for the SQP algorithm.

To identify the initial working set, we apply a similar rule as described in Chapter 3.4.5.1, but utilize the perturbed optimal solution, i.e.,

$$\mathcal{A}_{x,i} = \begin{cases} \text{inactive}, & \frac{x_i^{perturbed} - x_{L,i}}{\max(\tau_p, \lambda_{x,i}^{perturbed})} > \tau_{active} \text{ and } \frac{x_{U,i} - x_i^{perturbed}}{\max(\tau_p, -\lambda_{x,i}^{perturbed})} > \tau_{active}, \\ \text{active above}, & x_{U,i} = x_{L,i} \text{ or } \tau_{active} \geq \frac{x_i^{perturbed} - x_{l,i}}{\max(\tau_p, \lambda_{x,i}^{perturbed})} > \frac{x_{u,i} - x_i^{perturbed}}{\max(\tau_p, -\lambda_{x,i}^{perturbed})}, \\ \text{active below}, & \text{otherwise}, \end{cases}$$

and

$$\mathcal{A}_{c,j} = \begin{cases} \text{inactive}, & \frac{c_j(x^{perturbed}) - c_{L,j}}{\max(\tau_p, \lambda_{c,j}^{perturbed})} > \tau_{active} \text{ and } \frac{c_{U,j} - c_j(x^{perturbed})}{\max(\tau_p, -\lambda_{c,j}^{perturbed})} > \tau_{active}, \\ \text{active above}, & c_{U,j} = c_{L,j} \text{ and } \lambda_{c,j}^{perturbed} \leq 0, \text{ or} \\ & \tau_{active} \geq \frac{c_j(x^{perturbed}) - c_{L,j}}{\max(\tau_p, \lambda_j^{perturbed})} > \frac{c_{U,j} - c_j(x^{perturbed})}{\max(\tau_p, -\lambda_j^{perturbed})}, \\ \text{active below}, & \text{otherwise}. \end{cases}$$

These working sets also serve as the initial input for the SQP algorithm.

The SQP algorithm then runs with QORE as the QP subsolver, using the same configuration as in Chapter 3.5.1. The detailed results are presented in Table B.2.

Out of the 697 instances, 644 instances terminated within the 180-second time limit, and 588 instances were solved to optimality. However, some instances encountered failures due to various reasons. A breakdown of the failures is as follows:

- 22 instances failed because the predicted reduction is non-positive. [9]
- One instance failed because the trust-region radius became too small. This instance is `arwhead`.
- 6 instances encountered failure because the penalty parameter became too large. [10].
- 6 instances failed because the QP solver exceeded the maximum number of iterations. These instances are `arglinc`, `core2`, `corkscrw`, `flosp2hm`, `gausselm` and `sreadin3`
- 16 instances terminated because of internal QP solver errors. [11]
- 3 instances failed due to the QP solver reporting the QP problem is unbounded [12]
- 2 instances failed due to exceeding the maximum number of iterates[13].

A notable observation is that for those solved instances, 90% of them take less than 3 iterations with a total number of QP iterations less than 60, demonstrating the effectiveness of warm-starting both on the SQP solver level and the QP solver level.

3.5.1.2. Using Ipopt as QP subsolver. In this chapter, we run RestartSQP on the CUTE instances, using Ipopt as the selected QP subsolver. Ipopt was compiled with MA27 linear solver and was configured with a tolerance of 1e-12 and an acceptable tolerance of

[9]These instances are `arglinb`, `avion2`, `blockqp1`, `blockqp3`, `blockqp5`, `deconvu`, `flosp2hl`, `flosp2th`, `flosp2tl`, `flosp2tm`, `hs002`,, `hs029`, `hs037`, `hs041`, `lch`, `makela3`, `manne`, `pentagon`, `probpenl`, `rk23`, `s368` and `steenbrf`
[10]These instances are `coshfun`, `discs`, `goffin`, `optmass`, `orthrds2`, and `smmpsf`
[11]These instances are `dallass`, `eigenb2`, `gouldqp2`, `gridnetf`, `haifam`, `hs108`, `hvycrash`, `kissing`, `minperm`, `ngone`, `reading2`, `sipow3`, `spanhyd`, `twirism1`, `vanderm1` and `vanderm3`
[12]These instances are `brainpc7`, `reading1`, and `svanberg`.
[13]These instances are `meyer3` and `scosine`

1e-8. Additionally, we set `bound_relax_factor` in Ipopt option to 0 and we disabled the scaling of the QP by setting option value `nlp_scaling_method` to "none".

We chose this specific setup to test RestartSQP since it will be used in the two-stage optimization algorithm described in the next chapter. Currently, Ipopt and RestartSQP are the only dependencies for the two-stage algorithm, and therefore, we aim to ensure its reliability and performance under this configuration for future applications.

Using Ipopt as the QP subsolver, we obtained successful solutions for 696 instances within the specified time limit. Out of these, 640 instances were solved to optimality. The detailed statistics can be found in Table B.3. However, it is important to note that some instances did not converge to the optimal solution. Here is a breakdown of the failures:

- 12 instances failed due to exceeding the maximum number of iterates. [14].
- 5 instances failed because the predicted reduction is non-positive. [15].
- The instance `core1` failed because the trust-region region became too small.
- 13 instances encountered failure because the penalty parameter became too large. [16].
- 15 instances failed because the QP solver exceeded the maximum number of iterations. [17].
- 2 instances failed due to Ipopt throwing an error saying the QP problem formulation is invalid, [18]

[14]These instances are `chemrctb`, `cresc50`, `discs`, `himmelbd`, `hs063`, `orthrds2`, `orthrege`, `osbornea`, `palmer5a`, `palmer5e`, `palmer7a`, and `steenbrc`
[15]These instances are `hs013`, `launch`, `nonmsqrt`, `palmer2`, and `steenbre`
[16]These instances are `artif`, `britgas`, `fletcher`, `hs093`, `huestis`, `lakes`, `optctrl3`, `optctrl6`, `s365mod`, `ssebnln`, `vanderm2`, `vanderm3`, and `vanderm4`
[17]These instances are `core2`, `flosp2hh`, `flosp2hm`, `flosp2th`, `flosp2tm`, `meyer3`, `orthrgdm`, `orthrgds`, `palmer1c`, `palmer2e`, `scosine`, `scurly10`, `sinquad`, `steenbrb`, and `vardim`
[18]These instances are `cantilvr`, `semicon1`

- 6 instances failed due to the failure of Ipopt's restoration phase. [19].
- 2 instances failed due to errors in Ipopts step computation. [20].

We have observed that Ipopt performs better as a QP subsolver, just as anticipated. Its robustness contributes to the overall performance of the SQP solver as well.

3.6. Conclusing Remarks

In this chapter, we presented the development of a new C++ implementation of the SQP method. This implementation features the utilization of parametric QP methods and can efficiently handle a sequence of NLPs or crossover from the interior-point method. The practical performance of the RestartSQP software was demonstrated using the CUTE test set, and we conducted warm-start experiments to evaluate its effectiveness.

By incorporating warm-starting techniques at both the SQP solver level and the QP solver level, we can leverage the progress made in previous NLPs to accelerate convergence and significantly reduce the computational effort required for subsequent optimization problems. The warm-start approach has shown promising results in achieving rapid convergence. However, it is worth noting that the parametric active-set method utilized in the implementation may not be suitable for solving large-scale QP problems. Additionally, the QP solver QORE is still under development and lacks mature error handling compared to well-established solvers like Ipopt.

In the upcoming chapter, we will use RestartSQP as an integral part of the implementation for the two-stage optimization algorithm. This will allow us to showcase the full potential of the software in solving complex optimization problems in practical applications.

[19]These instances are `ncvxqp2`, `ncvxqp7`, `ncvxqp8`, `sawpath`, `smbank`, and `spanhyd`
[20]These instances are `steenbrf`, `twirism1`

CHAPTER 4

A Decomposition Algorithm for Continuous Nonlinear Two-Stage Optimization Problems

4.1. Introduction

In this chapter, we discuss a novel decomposition algorithm for nonlinear continuous two-stage problems, where the first-stage problem is given as

$$\min_{x_i \in \mathbb{R}^{n_i^P}, z \in \mathbb{R}^{n_0}} f_0(z) + \sum_{i=1}^{N} \hat{f}_i(x_i) \tag{4.1a}$$

$$\text{s.t } c_0(z) = 0, \tag{4.1b}$$

$$d_0(z) \leq 0, \tag{4.1c}$$

$$\hat{P}_i z - x_i = 0, \quad i = 1, \ldots, N, \tag{4.1d}$$

together with N ($N \geq 0$) second-stage problems of the form

$$\hat{f}_i(x_i) = \min_{y_i \in \mathbb{R}^{n_i}} f_i(y_i) \tag{4.2a}$$

$$\text{s.t } c_i(y_i) = 0, \tag{4.2b}$$

$$d_i(y_i) \leq 0, \tag{4.2c}$$

$$P_i y_i - x_i = 0. \tag{4.2d}$$

Here, $f_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}$, $c_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}^{m_i^c}$, and $d_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}^{m_i^d}$ are assumed to be twice-continuously differentiable. The shared variables $x_i \in \mathbb{R}^{n_i^P}$ are subvectors of the first-stage

variables z, formally defined as projections in (4.1d), where the $n_i^P \times n_0$-dimensional projection matrices $\hat{P}_i$ contain rows of the identity matrix. Similarly, the second-stage variables y_i in (4.2d) is defined by $n_i^P \times n_i$-dimensional projection matrices P_i. In this chapter, we will also refer to the first-stage problem as the master problem and the second-stage problems as the subproblems.

Instances like these often arise from the sample-average approximation of stochastic two-stage problems [**4, 52, 65, 9**], in which a first-stage decision has to be made before uncertain data in the second stage is revealed. In this setting, once the unknown data is realized, the second-stage problem determines a recourse action to minimize the corresponding second-stage cost. The goal then is to minimize the overall expected costs, where the randomness is approximated by N scenarios.

In some cases, networks with specific structures also give rise to two-stage formulations. For one of the applications considered in our numerical experiments, an electrical power grid is decomposed into a high-voltage transmission network (the first-stage problem) and a set of distribution networks (the second-stage problems) connected to the transmission grid at specific buses [**61, 77, 76**].

In contrast to approaches like Benders' algorithm for linear instances [**6**], the new method does not construct a first-stage problem that incorporates information from the second-stage problems by means of additional constraints (cuts) that are added sequentially to the first-stage problem. Instead, the proposed method solves the first-stage problem (4.1) directly, keeping the value functions $\hat{f}_i$ as part of the objective function.

A distinguishing strength of the new approach is that it can utilize standard nonlinear optimization solvers that handle the nonlinearity of the instances in a natural way. To make this work, however, we need to address a challenge that makes the direct application of a

fast second-order nonlinear optimization solver for the first-stage problems impossible: The second-stage value function $\hat{f}_i$ is typically not differentiable at values of x_i when the set of inequality constraints (4.2c) that are active at the optimal solution changes as x_i is varied. To overcome this predicament, the proposed algorithm relies on a smoothing approach that results in a smooth approximation of the value function $\hat{f}_i$ so that a standard nonlinear optimization method can be used to solve the smoothed first-stage problem. The smoothing technique is described in Chapter 4.3.

4.1.1. Related work

Two-stage optimization decomposition algorithms have become increasingly popular in the field of optimization for addressing large-scale problems with a hierarchical structure. In particular, stochastic programs pose significant challenges due to their exponential growth in size and complexity as the number of scenarios increases. To tackle these difficulties, decomposition algorithms such as Benders decomposition [**6, 78**] have been developed to solve linear programming and mixed-integer linear programming stochastic problems.

Benders decomposition, initially introduced for linear problems, was later generalized by Geoffrion [**32**] to handle nonlinear two-stage problems. This algorithm iteratively constructs a linear approximation of the master problem. At each iteration, the master problem is solved to obtain a feasible solution for the first-stage variables. Then, the subproblems are solved with the fixed first-stage solution, generating information to enhance the master problem's objective value. This information typically takes the form of linearized cuts or dual information. The algorithm continues this iterative process until convergence, usually determined by a stopping criterion such as the duality gap or a specified number of iterations. However, a drawback of Benders decomposition is that the master problem may contain a

large number of constraints, resulting in the repetitive solving of subproblems.

To overcome the limitations of Bender's decomposition method, which requires convexity, gradient-based two-stage decomposition algorithms have been developed. One notable decomposition framework in this category is Collaborative Optimization (CO), introduced by Braun [**13**]. The algorithm employs an inexact penalty function to decompose the problem and create independent subproblems by introducing copies of global variables. However, both the CO master problem and subproblems suffer from degeneracy [**19**].

To address this issue, DeMiguel and Murray [**20**] proposed alternative formulations and algorithms. The Inexact Penalty Decomposition algorithm utilizes an inexact penalty function, while the Exact Penalty Decomposition algorithm combines an exact penalty function with a barrier term. Both algorithms have been shown to achieve local convergence at a superlinear rate under mild assumptions. It is worth noting that tuning the penalty parameter in the exact penalty formulation can be challenging [**14**]. Other methods for addressing CO include [**72, 42**].

Our algorithm builds upon the work proposed by [**77**], which introduced the initial formulation and methodology. The smoothing technique utilized in our algorithm has been adapted from the broader field of general bilevel programming [**71, 74, 48**]. It is important to note that the two-stage optimization problem can be viewed as a special case of bilevel programming. Furthermore, our work shares similarities with [**1, 73**], where the subproblems are smoothed using a barrier term.

4.1.2. Overview

Chapter 4.3 introduces the smoothing technique that renders the second-stage recourse function differentiable, and discussed the computation of derivatives for the first-stage problem.

The overall decomposition framework is then described in Chapter 4.4. Chapter 4.6 gives a detailed description of our implementation, followed by extensive numerical experiments in Chapter 4.7.

4.2. Preliminaries

4.2.1. Full recourse property

For the most part of the paper, we make the following assumption on the second-stage problems:

Assumption 4.2.1. *The second-stage problem has the full recourse property, that is, it is feasible for any value of x_i.*

However, to relax this assumption, the implemented algorithm considers a penalty-based second-stage problem defined as

$$\hat{f}_i(x_i;\rho) = \min_{y_i,t_i,w_i} f_i(y_i) + \rho \sum_j (t_{ij} + w_{ij}) \tag{4.3a}$$

$$\text{s.t } c_i(y_i) = 0, \tag{4.3b}$$

$$d_i(y_i) \leq 0, \tag{4.3c}$$

$$P_i y_i - x_i + t_i - w_i = 0 \tag{4.3d}$$

$$t_i, w_i \geq 0. \tag{4.3e}$$

In this formulation, the coupling constraint (4.2d) can be violated, but at the price that the ℓ_1-norm of the violation, $\|P_i y_i - x_i\|_1$, multiplied by a positive penalty parameter $\rho > 0$, is added to the objective function. Then, if there exists a particular value of x_i for which the original second-stage problem is feasible, the relaxed formulation is feasible for any x_i, i.e., it has the strong recourse property, and the algorithm described in the coming chapters can be applied to the relaxed formulation.

Under certain standard regularity assumptions, it can be shown that solving the relaxed formulation with a sufficiently large penalty parameter ρ can yield an optimal solution to the original unrelaxed problem. Specifically, if ρ is greater than the optimal multiplier of (4.3d), then the relaxed formulation can capture an optimal solution. However, determining the appropriate value of ρ in advance is not feasible as it depends on the specific problem instance and its characteristics. To address this, updating rules can be developed to dynamically adjust the value of ρ during the solution process.

4.2.2. Two-stage problem

The two-stage problem, as defined by equations (4.1) and (4.2), can be reformulated as an undecomposed problem given by:

$$\min_{x,y,z} \quad f_0(z) + \sum_{i=1}^{N} f_i(y_i) \tag{4.4a}$$

$$\text{s.t.} \quad c_0(z) = 0, \qquad (\xi_0) \tag{4.4b}$$

$$d_0(z) \leq 0, \qquad (\lambda_0) \tag{4.4c}$$

$$\hat{P}_i z - x_i = 0, \qquad (\hat{\eta}_i) \quad \forall i \in [N], \tag{4.4d}$$

$$c_i(y_i) = 0, \qquad (\xi_i) \quad \forall i \in [N], \tag{4.4e}$$

$$d_i(y_i) \leq 0, \qquad (\lambda_i) \quad \forall i \in [N], \tag{4.4f}$$

$$P_i y_i - x_i = 0, \qquad (\eta_i) \quad \forall i \in [N]. \tag{4.4g}$$

In the above formulation, the Lagrange multipliers associated with constraints (4.4b)–(4.4g) are denoted as $(\xi_0, \lambda_0, \{\hat{\eta}_i\}_{i=1}^N, \{\xi_i\}_{i=1}^N, \{\lambda_i\}_{i=1}^N, \{\eta_i\}_{i=1}^N)$. To simplify notation, we use ξ to represent $(\xi_0, \xi_1, \cdots, x_N)$ and λ to represent $(\lambda_0, \lambda_1, \cdots, \lambda_N)$. Additionally, $\hat{\eta}$ represents $(\hat{\eta}_1, \cdots, \hat{\eta}_N)$, and η represents $(\eta_1, \cdots, \eta_N)$.

Throughout this chapter, we assume that the following assumptions hold for the undecomposed problem (4.4):

Assumption 4.2.2. *There exists a local solution (z^*, x^*, y^*) to the undecomposed problem (4.4) with Lagrange multipliers $(\xi^*, \lambda^*, \hat{\eta}^*, \eta^*)$ at which LICQ and the following KKT conditions hold:*

$$
\begin{aligned}
\nabla_z f_0(z^*) + \nabla_z c_0(z^*)\xi_0^* + \nabla_z d_0(z^*)\lambda_0^* + \sum_{i=1}^{N} \hat{P}_i^T \hat{\eta}_i^* &= 0, \\
\nabla_{y_i} f_i(y_i^*) + \nabla_{y_i} c_i(y_i^*)\xi_i^* + \nabla_{y_i} d_i(y_i^*)\lambda_i^* + P_i^T \eta_i^* &= 0, \quad i \in [N], \\
-\hat{\eta}_i^* - \eta_i^* &= 0, \quad i \in [N], \\
c_0(z^*) &= 0, \\
d_0(z^*) &\leq 0, \\
\hat{P}_i z^* - x_i^* &= 0, \quad i \in [N], \\
c_i(y_i^*) &= 0, \quad i \in [N], \\
d_i(y_i^*) &\leq 0, \quad i \in [N], \\
P_i y_i^* - x_i^* &= 0, \quad i \in [N], \\
\lambda_0^* &\geq 0, \\
\lambda_i^* &\geq 0, \quad i \in [N], \\
\lambda_0^* \circ d_0(z^*) &= 0, \\
\lambda_i^* \circ d_i(y_i^*) &= 0, \quad i \in [N].
\end{aligned}
\tag{4.5}
$$

Assumption 4.2.3. *Let* (z^*, x^*, y^*) *be a local solution with the corresponding Lagrange multipliers* $(\xi^*, \lambda^*, \hat{\eta}^*, \eta^*)$ *to the undecomposed problem* (4.4) *that satisfies the KKT conditions* (4.5)*, and strict complementarity holds.*

Assumption 4.2.4 (Second-order sufficient condition)**.** *Let* (z^*, x^*, y^*) *be a local solution with the corresponding Lagrange multipliers* $(\xi^*, \lambda^*, \hat{\eta}^*, \eta^*)$ *to the undecomposed problem* (4.4) *that satisfies the KKT conditions* (4.5)*, and the second-order sufficient condition holds at this local solution.*

4.2.3. First-stage problem optimality conditions

The first-order optimality conditions for the first-stage problem (4.1) are given by the following lemma.

Lemma 4.2.1. *Suppose* (z^*, x^*) *is a local solution to the first-stage problem* (4.1) *and that LICQ holds at* (z^*, x^*) *, then there exists Lagrange multipliers* $(\xi_0^*, \lambda_0^*, \hat{\eta}^*)$ *that satisfies the following conditions:*

$$
\begin{aligned}
\nabla_z \mathcal{L}_0(z^*, x^*, \xi_0^*, \lambda_0^*, \hat{\eta}^*) &= 0, \\
\nabla_{x_i} \mathcal{L}_0(z^*, x^*, \xi_0^*, \lambda_0^*, \hat{\eta}^*) &= 0, \quad i \in [N], \\
c_0(z^*) &= 0, \\
d_0(z^*) &\leq 0, \\
\hat{P}_i z^* - x_i^* &= 0, \quad i \in [N], \\
\lambda_0^* &\geq 0, \\
\lambda_0^* \circ d_0(z^*) &= 0,
\end{aligned}
\tag{4.6}
$$

where $\mathcal{L}_0(z^*, x^*, \xi_0^*, \lambda_0^*, \hat{\eta}_0^*)$ *is the Lagrangian function of the first-stage problem* (4.1)*, i.e.,*

$$
\mathcal{L}_0(z, x, \xi_0, \lambda_0, \hat{\eta}) = f_0(z) + c_0(z)^T \xi_0 + d_0(z)^T \lambda_0 + \sum_{i=1}^{N} (\hat{P}_i z - x_i)^T \hat{\eta}_i.
$$

Let F_0 be the vector function defined as

$$F_0(z, x, \xi_0, \lambda_0, \hat{\eta}) = \begin{bmatrix} \nabla_z \mathcal{L}_0(z, x, \xi_0, \lambda_0, \hat{\eta}) \\ \nabla_x \mathcal{L}_0(z, x, \xi_0, \lambda_0, \hat{\eta}) \\ c_0(z) \\ [d_0(z)]^+ \\ \hat{P}z - x \\ [-\lambda_0]^+ \\ \lambda_0 \circ d_0(z) \end{bmatrix}, \tag{4.7}$$

where $\hat{P}$ is a matrix that contains the rows $\hat{P}_i$ for $i \in [N]$. Then the KKT conditions (4.6) can be written in the following form as a set of nonlinear equations:

$$F_0(z^*, x^*, \xi_0^*, \lambda_0^*, \hat{\eta}^*) = 0.$$

4.3. Smoothing the Second-stage Problems

4.3.1. Barrier problem formulation

As discussed earlier, a direct application of a fast second-order nonlinear optimization solver to the first-stage problem is not practical, since the functions $\hat{f}_i(x_i)$ are not differentiable at points where there is a change in the set of second-stage constraints that are active at the optimal solution.

We give a simple example in Example 4.3.1 from [**76**] to illustrate this issue.

Example 4.3.1. *Consider the following two-stage problem*

$$\min_{x \in \mathbb{R}^2} \quad \hat{f}(x)$$

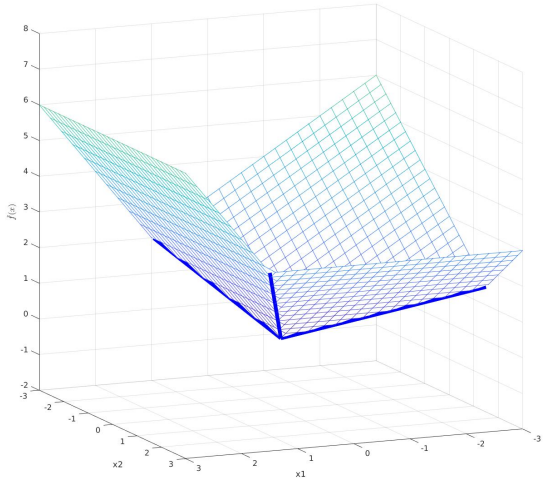


Figure 4.1. The function $\hat{f}(x)$ is not differentiable at $(x_1^*, x_2^*) = 0$.

with the second-stage problem given by

$$\begin{aligned} \hat{f}(x) = \min_{y \in \mathbb{R}} \quad & y \\ s.t \quad & y \geq 2x_1, \\ & y \geq x_2, \\ & y \geq -x_1 - x_2. \end{aligned}$$

The optimal solution is $(x_1^*, x_2^*, y^*) = 0$*, and the optimal value is* 0*. However, the function* $\hat{f}(x)$ *is not differentiable at* $(x_1^*, x_2^*) = 0$*.*

To overcome this dilemma, we replace the inequality constraints in the second-stage problem with log-barrier terms that are added to the objective function. This leads to the classical barrier problem formulation

$$\min_{y_i \in \mathbb{R}^{n_i}, s_i \in \mathbb{R}^{m_i^d}} f_i(y_i) - \mu \sum_{j=1}^{m_i^d} \log(s_{ij})$$

$$\text{s.t } c_i(y_i) = 0,$$

$$d_i(y_i) + s_i = 0,$$

$$P_i y_i - x_i = 0,$$

with barrier parameter $\mu > 0$ and slack variables s_i. We denote $\hat{f}_i(x_i; \mu)$ as the approximation of its optimal objective based on this barrier problem formulation. Since now there are no inequality constraints left, a change in the set of active inequality constraints can no longer lead to non-differentiability.

Let us assume, for the purpose of this discussion, that the second-stage problems are convex. This implies that c_i is affine, and both f_i and d_i are convex functions. Additionally, we assume that the optimal solutions to these problems are unique. It is a well-known result that as μ approaches zero, the optimal solution $y^*(x_i; \mu)$ of the approximation (4.8) converges to the optimal solution $y^*(x_i)$ of the original subproblem.

Based on this convergence behavior, we can construct an approximation $\hat{f}_i(x_i; \mu)$ of the original value function $\hat{f}_i(x_i)$ by utilizing the optimal solution $y^*(x_i; \mu)$ from (4.8). It is desirable for $\hat{f}_i(x_i; \mu)$ to be differentiable and converge to $\hat{f}_i(x_i^*)$ as μ goes to zero. In other words, we can interpret μ as a smoothing parameter that determines the closeness of the approximation $\hat{f}_i(x_i; \mu)$ to $\hat{f}_i(x_i^*)$.

This provides the basis for the decomposition algorithm described later in Chapter 4.4: We obtain an optimal solution of the original two-stage problem by solving a sequence of smoothed two-stage problems where μ is driven to zero.

For further analysis, we make the following assumptions on the second-stage problems. We assume that Assumption 4.2.1 holds for the original subproblem (4.2). It implies that the smoothed subproblem (4.8) is also feasible for any x_i and $\mu > 0$.

4.3.2. Natural approximation of the value function

With a fixed value of μ and utilizing the optimal solution $y^*(x_i;\mu)$ obtained from solving the smoothed subproblem (4.8), a natural approach to approximating the value function is to substitute $y^*(x_i;\mu)$ into the objective function of the original subproblem presented in equation (4.2). Hence, we define the approximation of the value function as follows:

$$\hat{f}_i(x_i;\mu) := f_i(y_i^*(x_i;\mu)). \tag{4.9}$$

The approximation of $\hat{f}_i$ stated above allows us to compute the gradient and Hessian information using the chain rule on the approximated objective $\hat{f}_i(x_i;\mu)$, i.e.,

$$\nabla_{x_i}\hat{f}_i(x_i;\mu) = \nabla_{x_i}y_i^*(x_i;\mu)\nabla_{y_i}f_i(y_i^*)$$

$$\nabla^2_{x_ix_i}\hat{f}_i(x_i;\mu) = \nabla_{x_i}y_i^*(x_i;\mu)\nabla^2_{y_iy_i}f_i(y_i^*)\nabla_{x_i}y_i^*(x_i)^T + \sum_{j=1}^{n_i}\frac{\partial f_i}{\partial y_{ij}}(y_i^*)\nabla^2_{x_ix_i}y_{ij}^*(x_i).$$

To compute the gradient and Hessian information for the approximated optimal value function, we need to calculate the quantities $\nabla_{x_i}y_i^*(x_i)$ and $\nabla^2_{x_ix_i}y_{ij}^*(x_i)$. These quantities can be obtained using the implicit function theorem.

4.3.2.1. Application of implicit function theorem. Before going into the detailed derivations, we need to make the following assumption:

Assumption 4.3.1. *The objective function f_i and constraints c_i, d_i are three times differentiable in y_i.*

Given Assumption 4.2.1, we let $(y_i^*, s_i^*) = (y_i^*(x_i;\mu), s_i^*(x_i;\mu))$ be the primal solution with corresponding Lagrange multipliers $(\xi_i^*, \lambda_i^*, \eta_i^*) = (\xi_i^*(x_i;\mu), \lambda_i^*(x_i;\mu), \eta_i^*(x_i;\mu))$. Then the following perturbed KKT conditions hold:

$$\begin{aligned}
\nabla_{y_i}\mathcal{L}_i(y_i^*, s_i^*, \xi_i^*, \lambda_i^*, \eta_i^*; x_i^*) &= 0,\\
c_i(y_i^*) &= 0,\\
d_i(y_i^*) + s_i &= 0,\\
P_i y_i^* - x_i &= 0,\\
s_i^* \circ \lambda_i^* - \mu e &= 0,
\end{aligned}$$

where $\mathcal{L}_i$ is the Lagrangian function of the subproblem (4.2), i.e,

$$\mathcal{L}_i(y_i, s_i, \xi_i, \lambda_i, \eta_i; x_i) = f_i(y_i) + c_i(y_i)^T\xi_i + (d_i(y_i) + s_i)^T\lambda_i + (P_i y_i - x_i)^T\eta_i.$$

Let $u_i^*(x_i) = (y_i^*(x_i), s_i^*(x_i), \xi_i^*(x_i), \lambda_i^*(x_i), \eta_i^*(x_i))$, the optimality conditions can be rewritten as

$$F_i(x_i, u_i^*(x_i); \mu) = \begin{bmatrix} \nabla_{y_i} \mathcal{L}_i(y_i^*, s_i^*, \xi_i^*, \lambda_i^*, \eta_i^*; x_i) \\ c_i(y_i^*) \\ d_i(y_i^*) + s_i^* \\ P_i y_i^* - x_i \\ s_i^* \circ \lambda_i^* - \mu e \end{bmatrix} = 0. \tag{4.11}$$

The application of the implicit function theorem to derive $\nabla_{x_i} y_i^*(x_i)$ and $\nabla^2_{x_i x_i} y_i^*(x_i)$ can be accomplished using (4.11). For the first-order derivative, we apply the chain rule directly to F_i and obtain

$$0 = \nabla_{x_i}(F_i(x_i, u_i^*(x_i))) = \nabla_{x_i} F(x_i, u_i^*(x_i)) + \nabla_{x_i} u_i^*(x_i) \nabla_{u_i} F(x_i, u_i^*(x_i)).$$

Here, $\nabla_{u_i} F_i^T$ denotes the Jacobian of F_i with respect to u_i, given by

$$\nabla_{u_i} F_i^T = \begin{bmatrix} \nabla^2_{y_i y_i} \mathcal{L}_i & 0 & \nabla_{y_i} c_i & \nabla_{y_i} d_i & P_i^T \\ \nabla_{y_i} c_i^T & 0 & 0 & 0 & 0 \\ \nabla_{y_i} d_i^T & I & 0 & 0 & 0 \\ P_i & 0 & 0 & 0 & 0 \\ 0 & \Lambda_i & 0 & S_i & 0 \end{bmatrix}, \tag{4.12}$$

where $\Lambda_i = \text{diag}(\lambda_i^*)$, $S_i = \text{diag}(s_i^*)$. To obtain $\nabla_{x_i} u_i^*(x_i)$, we solve the following linear system:

$$\nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \nabla_{x_i} u_i^*(x_i)^T = -\nabla_{x_i} F_i(x_i, u_i^*(x_i))^T. \tag{4.13}$$

Given that F_i is linear in x_i, the right-side of (4.13) is given by

$$-\nabla_{x_i} F_i = \begin{bmatrix} 0 & 0 & 0 & I_{n_i^P} & 0 \end{bmatrix}.$$

It indicates that $\nabla_{x_i} u_i^*$ is a submatrix of $\nabla_{u_i} F_i^{-T}$ that corresponds to the variables η_i. It should be noted that $\nabla_{x_i} u_i^* = (\nabla_{x_i} y_i^*, \nabla_{x_i} s_i^*, \nabla_{x_i} \xi_i^*, \nabla_{x_i} \lambda_i^*, \nabla_{x_i} \eta_i^*) \in \mathbb{R}^{n_i^P \times K_i}$, where $K_i = n_i + m_i^c + 2m_i^d + n_i^P$ is the total number of rows of F_i. Solving (4.13) for $\nabla_{x_i} u_i^*$ requires solving a linear system n_i^P times. Finally, the desired quantity $\nabla_{x_i} y_i^*$ is a submatrix of the first n_i rows of $\nabla_{x_i} u_i^*$.

For deriving second-order derivatives, we apply the chain rule again. By fixing x_{ij} and x_{ik}, and using the fact that F_i is linear in x_i we have

$$\begin{aligned} 0 &= \frac{\partial^2}{\partial x_{ij} \partial x_{ik}} F_i(x_i, u_i^*(x_i)) \\ &= \frac{\partial}{\partial x_{ik}} \left[\frac{\partial F_i}{\partial x_{ij}}(x_i, u_i^*(x_i)) + \nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial u_i^*}{\partial x_{ij}}(x_i) \right] \\ &= \frac{\partial}{\partial x_{ik}} \left[\nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial u_i^*}{\partial x_{ij}}(x_i) \right] \end{aligned} \tag{4.14}$$

where

$$\begin{aligned} &\frac{\partial}{\partial x_{ik}} \left[\nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial u_i^*}{\partial x_{ij}}(x) \right] \\ =&\Big(\frac{\partial}{\partial x_{ik}} \nabla_{u_i} F_i^T(x_i, u_i^*(x_i)) \Big) \frac{\partial u_i^*}{\partial x_{ij}}(x) + \nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial^2 u_i^*}{\partial x_{ik} \partial x_{ij}}(x) \\ =&\frac{\partial \nabla_{u_i} F_i^T}{\partial x_{ik}}(x_i, u_i^*(x_i)) \frac{\partial u_i^*}{\partial x_{ij}}(x_i) + \frac{\partial u_i^*}{\partial x_{ij}}(x_i) \nabla^2_{u_i u_i} F_i(x_i, u_i^*(x_i)) \frac{\partial u_i^*}{\partial x_{ik}}(x) + \\ &\nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial^2 u_i^*}{\partial x_{ik} \partial x_{ij}}(x_i). \end{aligned}$$

with $\frac{\partial u_i^*}{\partial x_{ij}}(x_i)^T \nabla^2_{u_i u_i} F(x, u(x_i)) \frac{\partial u_i^*}{\partial x_{ik}}(x_i)$ defined as

(4.15)

$$\frac{\partial u_i^*}{\partial x_{ij}}(x_i)^T \nabla^2_{u_i u_i} F_i(x_i, u_i^*(x_i)) \frac{\partial u_i^*}{\partial x_{ik}}(x_i) = \begin{bmatrix} (\frac{\partial u_i^*}{\partial x_{ij}}(x_i))^T \nabla^2_{u_i u_i} F_{i,1}(x_i, u_i^*(x_i))(\frac{\partial u_i^*}{\partial x_{ik}}(x_i)) \\ \vdots \\ (\frac{\partial u_i^*}{\partial x_{ij}}(x_i))^T \nabla^2_{u_i u_i} F_{i,K_i}(x_i, u_i^*(x_i))(\frac{\partial u_i^*}{\partial x_{ik}}(x_i)) \end{bmatrix}.$$

We further note that since $\nabla_{u_i} F_i^T$ given in (4.12) is independent of x_i, we have $\frac{\partial \nabla_{u_i} F_i^T}{\partial x_{ik}}(x_i, u_i^*(x_i)) = 0$. From this, we reduce (4.14) to

$$0 = \frac{\partial u_i^*}{\partial x_{ij}}(x_i) \nabla^2_{u_i u_i} F_i(x_i, u_i^*(x_i)) \frac{\partial u_i^*}{\partial x_{ik}}(x) + \nabla_{u_i} F_i(x_i, u_i^*(x_i))^T \frac{\partial^2 u_i^*}{\partial x_{ik} \partial x_{ij}}(x_i).$$

The desired quantity $\frac{\partial^2 u_i^*}{\partial x_{ik} \partial x_{ij}}(x_i)$ can be then calculated from

$$\frac{\partial^2 u_i^*}{\partial x_{ij} \partial x_{ik}} = -\nabla_{u_i} F_i^{-T} \Big[\frac{\partial u_i^*}{\partial x_{ij}}(x_i)^T \nabla^2_{u_i u_i} F_i(x_i, u_i^*(x_i)) \frac{\partial u_i^*}{\partial x_{ik}}(x_i) \Big] \tag{4.16}$$

Here, the Jacobian term $\nabla_{u_i} F_i^T$ used in the expression only needs to be factorized once and can be reused in all calculations of both first-order and second-order derivatives.

Remark 4.3.1. *We want to note that for $l = 1, \ldots, n_i$, we have*

$$\nabla^2_{u_i u_i} F_{i,l} = \frac{\partial}{\partial y_{il}} (\nabla_{u_i} F_{i,l})^T \tag{4.17}$$

which involves computing the third-order derivative of $\mathcal{L}_i$ with respect to y. In other words, to calculate $\nabla^2_{u_i u_i} F_{i,l}$, it is necessary to determine the third-order derivatives of f_i, c_i, and d_i. For l corresponding to the condition $c_i(y_i) = 0$, we have

$$\nabla^2_{u_i u_i} F_{i,l} = \begin{bmatrix} \nabla^2_{y_i y_i} c_{i,l}(y_i) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Similarly, for l corresponding to the condition $d_i(y_i) + s_i = 0$, we have

$$\nabla^2_{u_i u_i} F_{i,l} = \begin{bmatrix} \nabla^2_{y_i y_i} d_{i,l}(y_i) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

For l corresponding to the condition $P_i y_i = x_i$, we have $\nabla^2_{u_i u_i} F_{i,l} = 0$. Finally, for l corresponding to the condition $s_i \circ \lambda_i - \mu e = 0$, we have

$$\nabla^2_{u_i u_i} F_{i,l} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \operatorname{diag}(e_l) & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \operatorname{diag}(e_l) & 0 & 0 & 0 \end{bmatrix},$$

where e_l is the l-th unit vector.

4.3.2.2. Limitations of the natural approximation. The natural approximation discussed in the previous chapter has demonstrated effectiveness in various cases, such as its

successful application in solving two-stage ACOPF problems [**77**]. However, it is important to be aware of its limitations. Understanding these limitations can help identify scenarios where alternative approaches may be more suitable. The following limitations should be taken into consideration.

Firstly, the computational complexity of the natural approximation can be a significant limitation, particularly when dealing with a second-stage problem that involves nonlinear objectives f_i and constraints c_i, and d_i. Computing the second-order derivative of the Lagrangian function $\mathcal{L}_i$ from (4.17) requires evaluating the third-order derivatives of f_i, c_i, and d_i. However, evaluating third-order derivatives is often infeasible in popular optimization modeling languages such as AMPL [**30**] and JuMP [**53**], which only provide interfaces for second-order derivatives. Even when third-order derivatives are available, calculating the Hessian term remains computationally expensive. In particular, to compute the (i, j)-th term of $\nabla^2_{x_i x_i} u_i^*$ from (4.16), a set of K_i bilinear products defined in each entry of (4.15) is required. This is followed by solving a linear system specified in (4.16). This procedure needs to be repeated $O((n_i^P)^2)$ times. When the dimension of the connecting variables x_i is large (i.e., n_i^P is large), the computation can potentially demand a significant amount of memory.

Another limitation of the natural approximation is that it can introduce spurious nonconvexity. This can be seen in the following example.

Example 4.3.2. *Consider the following second-stage problem:*

$$
\begin{aligned}
\hat{f}(x_1, x_2) = \min_{y \in \mathbb{R}^{17}} \quad & 0.063y_1 + 0.048y_2 + 0.077y_3 + 0.054y_4 + 0.032y_5 + \\
& 0.077y_6 + 0.061y_7 + 0.093y_8 + 0.065y_9 + 0.020y_{10} + \\
& (y_{11} + y_{12} + y_{13} + y_{14} + y_{15}) + 0.1(y_{16} + y_{17}) \\
s.t. \quad & -x_1 + y_1 + y_2 + y_3 + y_4 + y_5 + y_{16} = 0, \\
& -x_2 + y_6 + y_7 + y_8 + y_9 + y_{10} + y_{17} = 0, \\
& y_3 + y_6 + y_{11} \geq 5.5, \\
& y_4 + y_7 + y_{12} \geq 5.7, \\
& y_5 + y_8 + y_{13} \geq 2.3, \\
& y_6 + y_9 + y_{14} \geq 6.2, \\
& y_7 + y_{10} + y_{15} \geq 1.3, \\
& y_i \geq 0, \quad i = 1, \ldots, 17.
\end{aligned}
$$

Example 4.3.2 is visualized in Figure 4.2, and the smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ is visualized in Figure 4.3. Although the second-stage subproblem is linear, the smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ is not convex in this case. The nonconvexity of the smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ is evident from Figure 4.3.

4.3.3. Log-barrier approximation of the value function

In the previous chapter, we discussed the approximation of the first-order derivatives of the objective function of the second-stage problem $\hat{f}_i$, utilizing chain rules and the implicit

Figure 4.2. Value function $\hat{f}(x_1, x_2)$ for Example 4.3.2

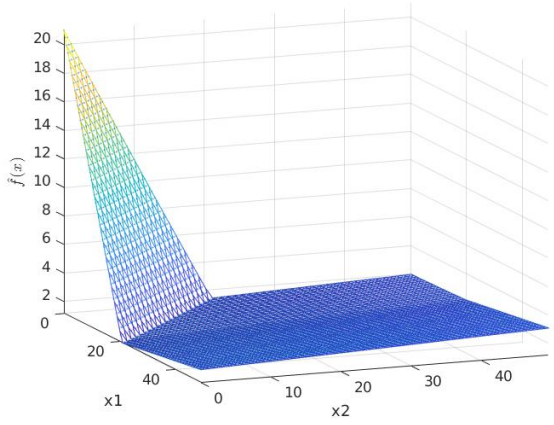
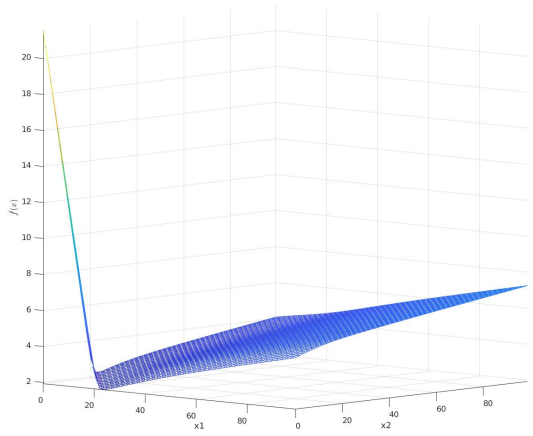


Figure 4.3. Smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ for Example 4.3.2 with natural approximation and $\mu = 0.1$



function theorem. Computing the gradients requires solving multiple linear systems, as shown in equation (4.13). Furthermore, computing the Hessian for $\hat{f}_i(x_i)$ is expensive, as it necessitates the third-order derivative of the Lagrangian function $\mathcal{L}_i$.

In this chapter, we will present an alternative approach to approximate $\hat{f}_i$. This approach considers adding the barrier term $-\mu \sum_j \log(s_{ij})$ into the approximated objective function,

which gives

$$\hat{f}_i(x_i;\mu) := f_i(y_i^*(x_i;\mu)) - \mu \sum_{j=1}^{n_i} \log(s_{ij}^*(x_i;\mu)). \tag{4.18}$$

In other words, $\hat{f}_i(x_i;\mu)$ represents the optimal objective function value of the barrier problem for a given vector x_i and parameter μ. Based on this formulation, we can establish the following lemma.

Lemma 4.3.1. *Suppose that (y_i^*, s_i^*) is a local optimal solution for the smoothed subproblem* (4.8) *together with the associated Lagrange multipliers $(\xi_i^*, \lambda_i^*, \eta_i^*)$ satisfying the second-order sufficient conditions, then*

$$\nabla_{x_i} \hat{f}_i(x_i;\mu) = -\eta_i^*(x_i) \tag{4.19}$$

and

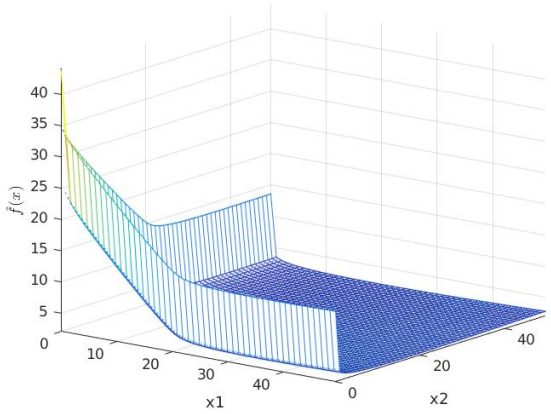
$$\nabla^2_{x_i x_i} \hat{f}_i(x_i;\mu) = -\nabla_{x_i} \eta_i^*(x_i)^T. \tag{4.20}$$

Proof. The equation (4.19) follows from [**54**, Chapter 11], and the equation (4.20) can be derived by taking the derivative on both sides of (4.19). □

The previous lemma demonstrates that the gradient and Hessian of the approximated objective function $\hat{f}_i(x_i;\mu)$ can be computed efficiently by retrieving the Lagrange multipliers $\eta_i^*(x_i)$ from the optimal primal-dual information of the smoothed subproblem (4.8). The Hessian term $-(\nabla_{x_i}\eta_i^*)^T$ corresponds to a submatrix of $-(\nabla_{x_i} u^*)^T$ and can be calculated using (4.13).

Compared to the natural approximation discussed in Chapter 4.3.2, the log-barrier approximation offers a more efficient Hessian calculation. While the natural approximation

Figure 4.4. Smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ for Example 4.3.2 with log-barrier approximation and $\mu = 0.1$



necessitates the computation of third derivatives for the objective function f_i and constraints c_i and d_i, the log-barrier approximation relies solely on their second derivatives. Moreover, since the Lagrange multipliers η_i^* can be obtained directly upon solving the smoothed subproblem (4.8), the computational cost mainly arises from solving (4.13). It is worth noting that even the simplest calculation of first-order derivatives using the natural approximation would require solving (4.13). Therefore, the log-barrier approximation offers computational advantages in terms of efficiency and simplicity in comparison to the natural approximation.

Furthermore, it is important to note that the log-barrier approximation represents a convex approximation of the smoothed second-stage objective function $\hat{f}_i(x_i; \mu)$. To illustrate this, Figure 4.4 depicts the visualized smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$ with the log-barrier approximation. As compared to Figure 4.3, the log-barrier approximation results in a convex representation of the smoothed second-stage objective function $\hat{f}(x_1, x_2; \mu)$.

Remark 4.3.2. *As shown in Figure 4.4, the log-barrier approximation exhibits a steep curvature as x_1 and x_2 approach 0, and it gets worse as μ goes to 0. This steep curvature can*

potentially introduce numerical challenges during the optimization process. In Chapter 4.6, we will address this issue and provide a detailed discussion of its implementation, including strategies to mitigate the numerical difficulties associated with the steep curvature of the log-barrier approximation.

4.4. Decomposition Framework

4.4.1. Perturbed two-stage formulation

This chapter presents the formal algorithm for solving the two-stage optimization problem given by equations (4.1) and (4.2). With the smoothed problem formulation given by equation (4.8), our framework solves the following perturbed two-stage problem:

$$\begin{aligned} \min\ & f_0(z) + \sum_{i=1}^{N} \hat{f}_i(x_i; \mu) \\ \text{s.t } & c_0(z) = 0, \\ & d_0(z) \leq 0, \\ & \hat{P}_i z - x_i = 0, \quad i \in [N]. \end{aligned} \tag{4.21}$$

Here, $\hat{f}_i(x_i; \mu)$ approximates $\hat{f}_i$ using either equation (4.9) or (4.18) for a sequence of barrier parameters $\{\mu^k\}$ such that $\mu^k \to 0$. As the barrier parameter μ^k goes to zero, the solution of the perturbed two-stage problems can recover that of the original problem.

4.4.2. Overall algorithm

The two-stage decomposition algorithm, as described in Algorithm 7, has been proven to converge, as discussed in [**76**]. The algorithm consists of outer and inner iterations. During the outer iteration, the objective is to solve the perturbed two-stage problems using a smoothing parameter μ^k. In each outer iteration, the perturbed two-stage problem is solved, and the smoothing parameter is updated iteratively until it becomes smaller than a specified tolerance level τ.

During each outer iteration, the perturbed two-stage problem is solved, and the smoothing parameter is decreased to obtain a new value μ^{k+1}. The smoothing parameter is updated iteratively until it is smaller than a specified tolerance level τ.

On the other hand, the inner iteration involves the use of a nonlinear optimization method to solve the first-stage problem. At each inner iteration l, we update the iterate $x_i^{k,l}$ according to the first-stage problem variable $z^{k,l}$. The subproblems are then solved using $x_i^{k,l}$ as a fixed parameter, and the gradient and/or Hessian information for $\hat{f}_i$ are computed. Using this information, we update the first-stage problem variable $z^{k,l+1}$. The inner iteration continues until the perturbed first-stage problem is solved to optimality.

Specifically, an NLP solver will be responsible for solving the first-stage problem until the KKT error for the first-stage problem is smaller than a specified tolerance τ_{master}, i.e., we want the iterate $(z^k, x^k, \xi_0^k, \lambda_0^k, \hat{\eta}^k)$ to satisfy $\|F_0(z, x, \xi_0, \lambda_0, \hat{\eta})\| \leq \tau_{master}(\mu^k)$, where F_0 is the KKT residual for the first-stage problem given by (4.7), and τ_{master} is a tolerance depends on μ^k. To solve the first-stage problem, one can employ a quasi-Newton method [**20**] or an SQP method for the subproblem, an interior-point method is typically preferred due to the smoothed reformulation (4.8).

During each iteration of the master NLP solver, the gradient and/or Hessian quantities for the second-stage value function $\hat{f}_i$ must be determined. As such, the subproblem solver is invoked whenever the master solver requires these quantities. In order to apply the implicit function theorem for gradient/Hessian calculations, as discussed in Chapter 4.3, it is essential to solve the subproblems to optimality. However, in practical implementations, the subproblems are typically solved up to a predefined tight tolerance level. For further information on this aspect, we recommend referring to Chapter 4.6.

Algorithm 7 Two-stage Decomposition Algorithm

Require: Initial smoothing parameter μ^0, initial iterate z^0, stopping tolerance τ, τ_{master} for the main algorithm and the first-stage problem solver respectively.

1: $k \leftarrow 0$
2: **while** $\mu^k > \tau$ **do**
3: Solve first-stage problem (4.21) with an NLP solver with initial iterate z^k and tolerance level $\tau_{master}(\mu^k)$ for z^{k+1}.
Whenever the NLP solver requires $\hat{f}_i$ or its derivatives, solve the corresponding subproblem (4.8) with x_i^k, μ^k and calculate the required quantities.
4: Decrease μ^k for μ^{k+1}.
5: $k \leftarrow k + 1$.
6: **end while**

4.5. Extrapolation Steps

In this chapter, we discuss the selection of starting points in Algorithm 7 when the penalty parameter changes from μ^k to μ^{k+1}. The goal is to choose a point that is as close as possible to a stationary point $u(\mu^{k+1})$. To achieve this, we explore an approach proposed by [**40**] that employs a composite extrapolation step, resulting in a fast local convergence rate. Specifically, we will explain how to derive the extrapolation step in a two-stage setup, building upon the work of [**40**]. Before we proceed, we first introduce the following assumptions.

Assumption 4.5.1. *The first-stage problem* (4.1) *only contains equality constraints. That is*

$$\begin{aligned} \min_{x,z} \quad & f_0(z) + \sum_{i=1}^{N} \hat{f}_i(x_i; \mu) \\ s.t. \quad & c_0(z) = 0, \\ & \hat{P}_i z - x_i = 0, \quad \forall i \in [N]. \end{aligned}$$

With this assumption, we define the smoothed undecomposed problem as follows.

$$\min_{x,y,z,s} \quad f_0(z) + \sum_{i=1}^{N} f_i(y_i) - \mu \sum_{j} \log(s_{ij}) \tag{4.22a}$$

$$\text{s.t.} \quad c_0(z) = 0, \qquad (\xi_0) \tag{4.22b}$$

$$\hat{P}_i z - x_i = 0, \qquad (\hat{\eta}_i) \quad i \in [N], \tag{4.22c}$$

$$c_i(y_i) = 0, \qquad (\xi_i) \quad i \in [N], \tag{4.22d}$$

$$d_i(y_i) + s_i = 0, \qquad (\lambda_i) \quad i \in [N], \tag{4.22e}$$

$$P_i y_i - x_i = 0, \qquad (\eta_i) \quad i \in [N]. \tag{4.22f}$$

Remark 4.5.1. *Assumption 4.5.1 can be satisfied by identifying the active constraints in the first-stage problem* (4.1)*. When utilizing an SQP method to solve the first-stage problem, it automatically detects the optimal active set in proximity to the solution, provided that LICQ holds and the strict complementarity condition are satisfied.*

Let (z^*, x^*, y^*) the optimal solution of the problem (4.22), the first-order optimality conditions of the problem (4.22) are given by

$$
\begin{aligned}
\nabla_z f_0(z^*) + \nabla_z c_0(z^*)\xi_0^* + \sum_{i=1}^{N} \hat{P}_i^T \hat{\eta}_i^* &= 0, \\
-\hat{\eta}_i^* - \eta_i^* &= 0, \quad \forall i \in [N], \\
c_0(z^*) &= 0, \\
\hat{P}_i z^* - x_i^* &= 0, \quad \forall i \in [N], \\
\nabla_{y_i} f_i(y_i^*) + \nabla_{y_i} c_i(y_i^*)\xi_i^* + \nabla_{y_i} d_i(y_i^*)\lambda_i^* + P_i^T \eta_i^* &= 0, \quad \forall i \in [N], \\
c_i(y_i^*) &= 0, \quad \forall i \in [N], \\
d_i(y_i^*) + s_i^* &= 0, \quad \forall i \in [N], \\
P_i y_i^* - x_i^* &= 0, \quad \forall i \in [N], \\
s_i \circ \lambda_i^* - \mu e &= 0, \quad \forall i \in [N],
\end{aligned}
\tag{4.23}
$$

where $\xi_0^*, \hat{\eta}_i^*$ are the optimal Lagrange multipliers for the constraints (4.22b) and (4.22c), respectively, and $\xi_i^*, \lambda_i^*, \eta_i^*$ are the optimal Lagrange multipliers for the constraints (4.22d), (4.22e) and (4.22f), respectively.

Now define

$$
F_0(z, x, \xi_0, \hat{\eta}, \eta; \mu) = \begin{bmatrix} \nabla_z f_0(z) + \nabla_z c_0(z)\xi_0 + \hat{P}^T \hat{\eta} \\ -\hat{\eta} - \eta \\ \hat{P}z - x \\ c_0(z) \end{bmatrix},
$$

and

$$F_i(y_i, x_i, s_i, \xi_i, \eta_i, \lambda_i; \mu) = \begin{bmatrix} \nabla_{y_i} f_i(y_i) + \nabla_{y_i} c_i(y_i)\xi_i + \nabla_{y_i} d_i(y_i)\lambda_i + P_i^T \eta_i \\ c_i(y_i) \\ d_i(y_i) + s_i \\ P_i y_i - x_i \\ s_i \circ \lambda_i - \mu e \end{bmatrix}.$$

With the above definitions, the optimality conditions (4.23) can be written as

$$F(x, y, z, s, \xi, \lambda, \hat{\eta}, \eta; \mu) = \begin{bmatrix} F_0(z, x, \xi_0, \hat{\eta}, \eta; \mu) \\ F_1(y_1, x_1, s_1, \xi_1, \eta_1, \lambda_1; \mu) \\ \vdots \\ F_N(y_N, x_N, s_N, \xi_N, \eta_N, \lambda_N; \mu) \end{bmatrix} = 0. \tag{4.24}$$

Now let $u_0 = (z, x, \hat{\eta}, \xi_0), u_i = (y_i, s_i, \xi_i, \eta_i, \lambda_i)$ and $u = (u_0, u_1, \ldots, u_N)$, and further let $u^k = u(\mu^k)$ be the solution of $F(u; \mu^k) = 0$, and assume that μ^k is small enough, so that the Jacobian $\nabla_u F(u; \mu^k)^T$ is nonsingular and the implicit function yields that

$$\begin{aligned} \nabla_\mu u(\mu^k)^T &= -\nabla_u F(u(\mu^k); \mu^k)^{-T} \nabla_\mu F(u(\mu^k); \mu^k)^T \\ &= \begin{bmatrix} \nabla_{u_0} F_0^T & \nabla_{u_1} F_0^T & \cdots & \nabla_{u_N} F_0^T \\ \nabla_{u_0} F_1^T & \nabla_{u_1} F_1^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \nabla_{u_0} F_N^T & 0 & \cdots & \nabla_{u_N} F_N^T \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ e_1 \\ \vdots \\ e_N \end{bmatrix} \end{aligned}$$

where e_i is a vector of length equal to the number of rows of F_i with only the part corresponding to $s_i \circ \lambda_i - \mu e$ equals to ones, i.e., $e_i = (0\ 0\ 0\ 0\ e)^T$.

Using the first-order Taylor expansion of $u(\mu)$ about μ^k, we derive the extrapolation point as

$$u^{k,EX} = u(\mu^k) + \nabla_\mu u(\mu^k)^T(\mu^{k+1} - \mu^k).$$

The composite extrapolation step is then defined as combining a Newton step Δu solved by

$$\nabla_u F(u^k; \mu^{k+1})^T \Delta u^k = -F(u^k; \mu^{k+1})$$

and the extrapolation step, i.e.,

$$\begin{aligned} u^{k,PD} &= u^{k,EX} + \Delta u^k \\ &= u(\mu^k) + \nabla_\mu u(\mu^k)^T(\mu^{k+1} - \mu^k) - \nabla_u F(u(\mu^k); \mu^k)^{-T} F(u(\mu^k); \mu^k) \\ &= u^k + \nabla_u F(u^k; \mu^k)^{-T} \left(\begin{bmatrix} 0 \\ e_1 \\ \vdots \\ e_N \end{bmatrix} (\mu^{k+1} - \mu^k) - \begin{bmatrix} F_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right). \end{aligned}$$

The step Δu_0^k can be solved by the Schur complement method:

$$\left(\nabla_{u_0} F_0^T - \sum_{i=1}^{N} \nabla_{u_i} F_0^T (\nabla_{u_i} F_i)^{-T} \nabla_{u_0} F_i^T \right) \Delta u_0^k = -F_0 - \sum_{i=1}^{N} \nabla_{u_i} F_0^T \nabla_{u_i} F_i^{-T} e_i (\mu^{k+1} - \mu^k)$$

The implementation details for the extrapolation step are presented in Chapter 4.6.5.

4.6. Details of the Implementation

In this chapter, we present a detailed description of the implementation of the two-stage algorithm. The algorithm is implemented in C++, and it comes with interfaces in

C, AMPL [**30**], and JuMP [**53**]. The software relies on RestartSQP described in Chapter 3 for solving the master problem and Ipopt [**81**] for solving smoothed subproblems. The chapter is organized as follows. We first present the software design and discuss the reverse communication used in the software. Then, we discuss how we modify RestartSQP and Ipopt to solve the master problem and subproblems respectively.

4.6.1. Software design

This chapter presents the software design for the two-stage problem decomposition algorithm and provides an overview of how to set up and solve a problem. The software structure of the decomposition algorithm is illustrated in Figure 4.5. In this design, the solvers for the first-stage problem and subproblems operate as separate subroutines, each having its own optimization model.

To optimize the two-stage problem, a main driver program is necessary to coordinate the master solver and the subproblem solvers, facilitating the exchange of information between them. Algorithm 8 presents the pseudocode for the main driver program.

In the main driver program, the master solver, and subproblem solvers are initialized at the beginning. The first-stage optimization model is passed to the master solver, while each second-stage optimization model is passed to the respective second-stage subproblem solvers. Additionally, exchange data objects are created and maintained throughout the execution of the main driver program. The exchange data objects serve the purpose of transmitting updated first-stage variables x_i and the smoothing parameter μ to each subproblem solver. It is also responsible for sending optimal second-stage problem information to the master problem solver, including $\hat{f}_i$ as well as its gradient and Hessian information, once each subproblem is solved optimally. These exchange data structures enable communication between

the solvers, facilitating the transfer of relevant information. Status from the master solver, namely `master_status`, is shared among the solvers and is updated by the master problem solver to track the progress and determine the next step in the optimization process. This status is used to monitor the convergence or any other relevant information.

The main driver program follows an iterative process to solve the two-stage problem using the master solver and subproblem solvers. At the beginning of each iteration, the master solver is invoked to determine the next status. If gradient and Hessian information are required from the subproblems, the status is set to `solve_subproblems`. The master solver retains internal iteration information and returns the status to the main driver.

The main driver then shares relevant first-stage problem information, such as the current variable x_i^k and smoothing parameter μ^k, with each subproblem solver. It waits for the subproblem solvers to solve their respective subproblems. Within each subproblem solver, the gradient $\nabla_{x_i} \hat{f}_i$ and Hessian $\nabla^2_{x_i x_i} \hat{f}_i$ are calculated and stored in the exchange data.

The master problem solver uses the gradient and Hessian information obtained from the subproblem solvers to continue its iteration and update the status. If the master solver returns an optimal solution with the current smoothing parameter μ^k for a tolerance τ_{master}, then μ^k is updated by the master problem solver, and subsequent iterations are based on the new smoothing parameter μ^{k+1}. This process continues until the smoothing parameter becomes small enough, indicating that the first-stage problem has been solved to the requested overall tolerance τ.

It is important to highlight that the SQP algorithm described in Chapter 3 has been modified to use the reverse communication protocol when employed as the master solver in this two-stage algorithm. By utilizing reverse communication, the SQP function returns control to the caller whenever quantities such as derivative information are required but

all internal intermediate information of the SQP solver can be stored. Additionally, the subproblem solution processes can be run in parallel, taking advantage of the independence of each subproblem solver as an individual subroutine.

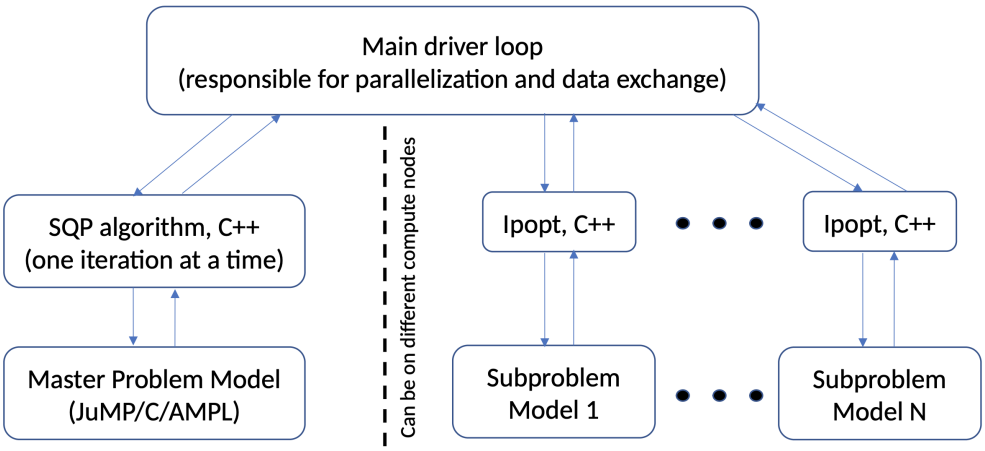


Figure 4.5. Decomposition algorithm software structure

Algorithm 8 Main driver program

Initialize master program solver and provide it with master problem model.
Initialize each subproblem solver and provide it with its subproblem model.
Initialize exchange data.
while True **do**
    Call master problem solver with exchange data to compute a new trial point. Upon return, the master solver will modify the exchange data and return `master_status`.
    if `master_status` does not require the solution of subproblems **then**
        break
    end if
    Call subproblem solvers to solve all subproblems and calculate required information. Upon return, each subproblem solver will update the exchange data.
end while
Get current primal-dual iterates from master problem solver and subproblem solvers.
return `master_status`, primal-dual iterates

4.6.2. Interfaces

The software comes equipped with a C interface for both the first-stage problem and subproblem optimization models. Furthermore, the software has been integrated with two optimization modeling languages, AMPL[**30**] and JuMP [**53**]. This feature allows users to conveniently and effectively formulate and solve their optimization models using the syntax and tools provided by these languages, in conjunction with the algorithms and solvers implemented in our software.

4.6.3. Master solver

We employ the SQP algorithm discussed in Chapter 3 to tackle the master problem. To utilize the reverse communication mechanism outlined in Chapter 4.6.1, we have made specific modifications to Algorithm 4 in our implementations.

Firstly, we have introduced an option in the SQP solver to allow users to specify whether the solver should employ the reverse communication mechanism. When this option is set to false, the solver exclusively solves the master problem using the regular SQP algorithm.

On the other hand, if the option is set to true, the solver object will promptly return to the main driver without solving any QP subproblems. The primal variables requiring evaluation are explicitly stored in the exchange data structure and passed back to the main driver. The return status of the SQP solver is adjusted accordingly to indicate that the solver is awaiting the solutions of subproblems at the trial iterate. Once the main driver receives this return status, it invokes the subproblem solvers to solve all the subproblems and calculates the gradient and Hessian information for the trial iterate. These quantities are then stored in the exchange data structure and returned to the main driver, awaiting transmission to

the SQP solver. Once all subproblems are solved, and the gradient/Hessian information is obtained, the main driver calls the SQP solver again, providing the gradient/Hessian information obtained from the subproblem solvers. The SQP solver then proceeds to solve the QP subproblem and continues with the subsequent steps of the SQP algorithm. This cycle repeats whenever new gradient/Hessian information for a new point is required.

4.6.4. Subproblem solver

In the implementation, we utilized the open-source package Ipopt [**81**] for solving the subproblems defined in equation (4.3). In this chapter, we will provide insights into how Ipopt internally handles the subproblem and computes the gradient and Hessian information. We will describe the implementation details for both the natural approximation and the log barrier approximation. Finally, we discuss the heuristics we implemented for improving the numerical stability of the subproblem solver.

To simplify notation, we denote the vector y_i from (4.3) as y_i^{orig} and represent the triplet (y_i^{orig}, t_i, w_i) as simply y_i. We can rewrite problem (4.3) as follows:

$$\min_{y_i} \; f_i(y_i; \rho) \tag{4.25a}$$

$$\text{s.t } \tilde{c}_i(y_i) = 0, \tag{4.25b}$$

$$\tilde{P}_i y_i - x_i = 0, \tag{4.25c}$$

$$d_{L,i} \leq d_i(y_i) \leq d_{U,i}, \tag{4.25d}$$

$$y_{L,i} \leq y_i \leq y_{U,i}, \tag{4.25e}$$

where $f_i(y_i;\rho) = f_i(y_i^{orig}) + \rho\left(\sum_j t_{ij} + w_{ij}\right)$ is the penalized objective function, and $\tilde{P}_i = [P_i\ 0\ 0]$. The bounds $y_{L,i} \in [-\infty,\infty)^{n_i} \times \{0\}^{2n_i^P}$, $y_{U,i} \in (-\infty,\infty]^{n_i+2n_i^P}$, $d_{L,i} \in [-\infty,\infty)^{m_{d,i}}$, and $d_{U,i} \in (-\infty,\infty]^{m_{d,i}}$ are assumed to satisfy $y_{L,i} < y_{U,i}$ and $d_{L,i} < d_{U,i}$, where the strict inequality holds in every vector component.

In the internal structure of Ipopt, the subproblem (4.25) is reformulated as follows:

$$\begin{aligned}
&\min_{y_i,s_i}\ f_i(y_i;\rho)\\
&\text{s.t } c_i(y_i) = 0,\\
&\quad \tilde{P}_i y_i - x_i = 0,\\
&\quad d_i(y_i) - s_i = 0,\\
&\quad y_{L,i} \le y_i \le y_{U,i},\\
&\quad d_{L,i} \le s_i \le d_{U,i}.
\end{aligned}$$

We define the projection matrix $P_{y_L,i}$ so that $P_{y_L,i}\ y_{L,i}$ filters the finite elements of $y_{L,i}$ and the relationship $y_{L,i} \le y_i$ is equivalent to $R_{y_L,i}(y_i) := P_{y_L,i}(y_i - y_{L,i}) \ge 0$. These operators are similarly defined for the other bounds, e.g., $R_{d_U,i}(s_i) := P_{d_U,i}(d_{U,i} - s_i)$. We further define $P_{x_i} = (0_{m_i^c \times n_i^P}, I)$ and $c_i(y_i) = (\tilde{c}_i(y_i), \tilde{P}_i y_i)$, then the constraints (4.25c) and (4.25d) can be written jointly as $c_i(y_i) - P_{x_i} x_i = 0$.

With this notation, the smoothed subproblem with a fixed barrier parameter $\mu > 0$ will be in the form:

$$
\begin{aligned}
\min_{y_i, s_i} \; & f_i(y_i; \rho) - \mu\big(\sum \log(R_{y_L,i}(y_i)) + \sum \log(R_{y_U,i}(y_i)) + \\
& \sum \log(R_{d_L,i}(s_i)) + \sum \log(R_{d_U,i}(s_i)) \big) \\
\text{s.t } & c_i(y_i) - P_{x_i} x_i = 0, \\
& d_i(y_i) - s_i = 0.
\end{aligned} \tag{4.26}
$$

Here, the term $\sum \log(w)$ for $w \in \mathbb{R}^n$ denotes $\sum_{i=1}^{n} \log(w_i)$.

4.6.4.1. Modification to Ipopt. To solve the smoothed subproblem (4.26) and utilize the linear algebra subroutines in Ipopt, we introduced specific modifications to the existing Ipopt implementation. The following adjustments were implemented:

(1) We changed Ipopt's option to enable the solver to terminate at a specified barrier parameter μ. This modification allows us to solve the smoothed subproblem (4.26) with a fixed μ, as required by our approach.

(2) We enabled Ipopt's warm-start option for solving the second-stage problems. By keeping the ipopt object, we retained all internal information and data structures, including iterate values and matrix factorization. This feature allows us to utilize the solution from a subproblem with smoothing parameter μ^k as a warm start for solving the subsequent subproblem with smoothing parameter μ^{k+1}.

(3) We introduced an option to specify the minimum number of iterations to be executed by the interior-point method. This addition is crucial to ensure that the internal linear algebra subroutine in Ipopt possesses up-to-date factorization information. In some scenarios, when the interior-point method terminates after zero iterations and declares optimality, the linear algebra routine may not accurately reflect the

changes made to the current optimal solution. By enforcing at least one iteration, we guarantee that the linear algebra subroutine is appropriately updated and maintains consistency with the latest solution.

(4) We added a termination option based on search direction size in the interior-point method. Specifically, we terminate the interior-point method if the search direction size is below a specified threshold. This modification is for handling the ill-conditioned Hessians from the QP problem in the SQP algorithm. This option is only turned on for nonlinear problems.

4.6.4.2. Derivative Computations. In Chapter 4.3, we presented two techniques for approximating the optimal objective gradient information of the subproblem. The first method, described in Chapter 4.3.2, involves substituting the optimal solution $y_i^*(x_i;\mu)$ of (4.8) into the objective function f and employing the implicit function theorem in conjunction with the chain rule to obtain the derivative information. The second method, described in Chapter 4.3.3, utilizes a log-barrier approximation for smoothed subproblem objective. By that, the optimal multiplier and Jacobian information can be directly used for calculating the gradient and Hessian information for $\hat{f}_i$. This chapter addresses the computation of derivatives for (4.26) utilizing both of these methods. We explain how we adapt and employ the existing implementations in Ipopt to efficiently compute the derivative information.

For simplicity, we omit the index i in this chapter except for some dimensional notation. The problem that Ipopt is solving for given x, ρ and μ is then of the form

$$\hat{f}(x_i;\rho,\mu) = \min_{y,s}\ f(y;\rho) - \mu\big(\sum \log(R_{y_L}(y)) + \sum \log(R_{y_U}(y)) + \sum \log(R_{d_L}(s)) + \sum \log(R_{d_U}(s))\big)$$

$$\text{s.t } c(y) - P_x x = 0,$$

$$d(y) - s = 0.$$

Let $u = (y, s, \lambda_c, \lambda_d, z_L, z_U, v_L, v_U)$. When solving the subproblems, the interior-point algorithm implemented in Ipopt computes the Newton step for the following system of equations:

$$F(x,u;\rho,\mu) = \begin{bmatrix} \nabla_y f(y;\rho) + \nabla_y c(y)\lambda_c + \nabla_y d(y)\lambda_d - P_{y_L}^T z_L + P_{y_U}^T z_U \\ -\lambda_d - P_{d_L}^T v_L + P_{d_U}^T v_U \\ c(y) - P_x x \\ d(y) - s \\ z_L \circ R_{y_L}(y) - \mu e \\ z_U \circ R_{y_U}(y) - \mu e \\ v_L \circ R_{d_L}(s) - \mu e \\ v_U \circ R_{d_U}(s) - \mu e \end{bmatrix} = 0.$$

It computes the Jacobian of F and solves linear systems with the Jacobian. The Jacobian of F is

$$\nabla_u F(x,u;\rho,\mu)^T =$$

$$\begin{bmatrix}
\nabla^2_{yy}\mathcal{L}(u) & 0 & \nabla c(y) & \nabla d(y) & -P^T_{y_L} & P^T_{y_U} & 0 & 0 \\
0 & 0 & 0 & -I & 0 & 0 & -P^T_{d_L} & P^T_{d_U} \\
\nabla c(y)^T & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\nabla d(y)^T & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
Z_L P_{y_L} & 0 & 0 & 0 & P_{y_L} Y & 0 & 0 & 0 \\
Z_U P_{y_U} & 0 & 0 & 0 & 0 & -P_{y_U} Y & 0 & 0 \\
0 & V_L P_{d_L} & 0 & 0 & 0 & 0 & P_{d_L} S & 0 \\
0 & V_U P_{d_U} & 0 & 0 & 0 & 0 & 0 & -P_{d_U} S
\end{bmatrix},$$

which is stored as a sparse matrix in Ipopt.

When the solver returns optimal solution $u^*(x) = (y^*, s^*, \lambda_c^*, \lambda_d^*, z_L^*, z_U^*, v_L^*, v_U^*)$ for a fixed μ, we can compute the optimal objective gradient information $\nabla_x u^*$ based on the implicit function theorem (4.13). Specifically, given that the derivative of F with respect to the first-stage variables is

$$\nabla_x F(x,u) = \begin{bmatrix} 0 & 0 & -P_x^T & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$\nabla_x u^*$ then is obtained by calling Ipopt's internal linear system solver for the linear system of the following form:

$$\nabla_u F(x, u^*; \rho, \mu)^T \frac{\partial u^*}{\partial x_j} = \begin{bmatrix} 0 \\ 0 \\ -e_{m_i^c + j} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.28}$$

for $j = 1, \cdots, n_i^P$. The gradient information $(\nabla_x u^*)^T$ is then explicitly stored as

$$(\nabla_x u^*)^T = \begin{bmatrix} \frac{\partial u^*}{\partial x_1} & \cdots & \frac{\partial u^*}{\partial x_{n_i^P}} \end{bmatrix}$$

for later use in the computation of the Hessian. The required quantity $(\nabla_x y^*)^T$ is then the first $n_i + 2m_i^d$ rows of $(\nabla_x u^*)^T$. The gradient of the subproblem objective $\nabla_x \hat{f}_i$ is computed from the chain rule as

$$\nabla_x \hat{f} = \nabla_x y^* \nabla_y f(y; \rho).$$

Our implementation of the Hessian computation using implicit function theorem is limited to cases where the subproblem objective f_i is quadratic and the constraints are linear. As we discussed in Chapter 4.3.2, when dealing with a general nonlinear objective function f_i and constraints c_i and d_i, it is necessary to calculate a third derivative in order to compute the derivative of the Lagrangian function. However, obtaining third derivative information is challenging, as most mathematical optimization modeling languages and their interfaces

are not equipped to provide it. Moreover, even for quadratic constraints c_i and d_i, computing the Hessian requires evaluating the constraints Hessian $m_i^c + m_i^d$ times, which can be computationally expensive in terms of function evaluations. Therefore, implementation is restrictive when the second-order derivative information of the constraints is zero.

Recall that for calculating the Hessian of $\hat{f}_i$, we have the following formula from the chain rule:

$$\nabla^2_{xx}\hat{f}(x) = \nabla_x y^*(x)\nabla^2_{yy}f(y^*;\rho)\nabla_x y^*(x)^T + \sum_l \frac{\partial f}{\partial y_l}(y^*)\nabla^2_{xx}y_l^*(x).$$

In our implementation, we compute the (j,k) entries of $\nabla^2_{xx}\hat{f}$ from

$$[\nabla^2_{xx}\hat{f}_i]_{jk} = \left(\frac{\partial y^*}{\partial x_j}\right)^T \nabla^2_{yy}f(y^*;\rho)\frac{\partial y^*}{\partial x_k} + \left(\frac{\partial^2 y^*}{\partial x_j \partial x_k}\right)^T \nabla_y f(y)$$

for $j = 1, \cdots, n_i^P$ and $k = 1, \cdots, n_i^P$. The first term is computed using gradient information $\nabla_x y^*$ and performing a matrix-vector multiplication. The second term is computed by first solving the following linear system

$$\nabla_u F_i \frac{\partial^2 u^*}{\partial x_j \partial x_k} = -\Big[\frac{\partial u}{\partial x_j}(x)^T \nabla^2_{uu}F_i(x,u^*;\rho,\mu)\frac{\partial u}{\partial x_k}(x)\Big]. \tag{4.29}$$

The right hand side vector $\frac{\partial u}{\partial x_j}(x)^T\nabla^2_{uu}F(x,u^*;\rho,\mu)\frac{\partial u}{\partial x_k}(x)$ is defined as

$$\frac{\partial u}{\partial x_j}(x)^T\nabla^2_{uu}F(x,u^*;\rho,\mu)\frac{\partial u}{\partial x_k}(x) = \begin{bmatrix} (\frac{\partial u}{\partial x_j}(x))^T\nabla^2_{uu}F_1(x,u^*;\rho,\mu)(\frac{\partial u}{\partial x_k}(x)) \\ \vdots \\ (\frac{\partial u}{\partial x_j}(x))^T\nabla^2_{uu}F_K(x,u^*;\rho,\mu)(\frac{\partial u}{\partial x_k}(x)) \end{bmatrix},$$

where $F_i(x,u^*;\rho,\mu)$ is the i-th row of $F(x,u^*;\rho,\mu)$ and K is the total number of rows of $F(x,u^*;\rho,\mu)$.

With quadratic objective and linear constraints, we can calculate (4.15) by using the following formula:

$$\frac{\partial u}{\partial x_j}(x)^T \nabla^2_{uu} F(x, u^*; \rho, \mu) \frac{\partial u}{\partial x_k}(x) = \begin{bmatrix} (\frac{\partial u}{\partial x_j}(x))^T \nabla^2_{uu} F_1(x, u^*; \rho, \mu)(\frac{\partial u}{\partial x_k}(x)) \\ \vdots \\ (\frac{\partial u}{\partial x_j}(x))^T \nabla^2_{uu} F_K(x, u^*; \rho, \mu)(\frac{\partial u}{\partial x_k}(x)) \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \left(P_{y_L} \frac{\partial y}{\partial x_j}(x)\right) \circ \frac{\partial z_L}{\partial x_k}(x) + \left(P_{y_L} \frac{\partial y}{\partial x_k}(x)\right) \circ \frac{\partial z_L}{\partial x_j}(x) \\ \left(P_{y_U} \frac{\partial y}{\partial x_j}(x)\right) \circ \frac{\partial z_U}{\partial x_k}(x) + \left(P_{y_U} \frac{\partial y}{\partial x_k}(x)\right) \circ \frac{\partial z_U}{\partial x_j}(x) \\ \left(P_{d_L} \frac{\partial s}{\partial x_j}(x)\right) \circ \frac{\partial v_L}{\partial x_k}(x) + \left(P_{d_L} \frac{\partial s}{\partial x_k}(x)\right) \circ \frac{\partial v_L}{\partial x_j}(x) \\ \left(P_{d_U} \frac{\partial s}{\partial x_j}(x)\right) \circ \frac{\partial v_U}{\partial x_k}(x) + \left(P_{d_U} \frac{\partial s}{\partial x_k}(x)\right) \circ \frac{\partial v_U}{\partial x_j}(x) \end{bmatrix}.$$

Given the above expression, we solve the linear system (4.29) following these steps. First, we use the above expression to fill in the right-hand side of the system, employing the stored values of $\nabla_x u^*$. To compute these values, we extract subvectors of $\nabla_x u^*$ to perform element-wise products and vector additions. We then use the Ipopt internal linear solver to solve the linear system corresponding to (4.29). The result, $\frac{\partial^2 u^*}{\partial x_j \partial x_k}$, is a vector of length equal to that of u. It comprises a subvector consisting of the first n_i elements, represented as $\frac{\partial^2 y^*}{\partial x_j \partial x_k}$. We then calculate the dot product between the $\frac{\partial^2 y^*}{\partial x_j \partial x_k}$ and $\nabla_y f$ and add the product to $\left(\frac{\partial y^*}{\partial x_j}\right)^T \nabla^2_{yy} f(y^*; \rho) \frac{\partial y^*}{\partial x_k}$ to get the (j,k)-th entry of $\nabla^2_{xx} \hat{f}$. Note that this process is repeated for all $j = 1, \cdots, n_i^P$ and for all $k = 1, \cdots j$ given the symmetric structure of the Hessian.

Next, we delve into the detailed calculation procedure to compute the gradient and Hessian for the log-barrier approximation of $\hat{f}$. After the solver Ipopt solver Ipopt returns $u^*(x) = (y^*, s^*, \lambda_c^*, \lambda_d^*, z_L^*, z_U^*, v_L^*, v_U^*)$ as the optimal solution, we can determine the gradient $\nabla_x \hat{f}$ by identifying the relevant multiplier within λ_c^* and assigning the negative of that as $\nabla_x \hat{f}$. This approach allows us to obtain the gradient efficiently without additional function evaluations. To compute the second-order derivative, we calculate $(\nabla_x u^*)^T$ by solving the linear system (4.28) as discussed previously. This enables us to obtain the required information for the Hessian matrix. Specifically, we select the rows corresponding to P_x in $(\nabla \lambda_c^*)^T$ to obtain the submatrix that represents the Hessian.

Overall, this approach provides an efficient and accurate method for calculating the gradient and Hessian using the information obtained from the solver Ipopt. By leveraging the results from Ipopt and solving the linear system (4.28) using Ipopt's internal linear solver subroutines, we can avoid unnecessary function evaluations and achieve computational savings. Moreover, within the two-stage algorithm framework, there is no need to solve linear systems or perform symbolic factorization of the matrix since these tasks are already handled internally by Ipopt.

4.6.5. Extrapolation step calculation

In this chapter, we present the implementation details for extrapolation step calculation. For simplicity, we will consider the case where $N = 1$ and assume that the first-stage problem has no inequality constraints as before.

Consider the undecomposed problem (4.4). At an iterate (u_0^k, u_1^k), the primal-dual interior-point method for the undecomposed problem with a barrier parameter μ_l computes

a search direction $(\Delta u_0^k, \Delta u_1^k)$ by solving the linear system

$$\begin{bmatrix} \nabla_{u_0} F_0^T & \nabla_{u_1} F_0^T \\ \nabla_{u_0} F_1^T & \nabla_{u_1} F_1^T \end{bmatrix} \begin{bmatrix} \Delta u_0^k \\ \Delta u_1^k \end{bmatrix} = - \begin{bmatrix} F_0(u_0^k, u_1^k; \mu_l) \\ F_1(u_0^k, u_1^k; \mu_l) \end{bmatrix} \tag{4.30}$$

In our decomposition algorithm, we usually adjust the subproblem variable u_1^k so that $F_1(u_0^k, u_1^k; \mu_l) = 0$, but in order to mirror the fast local convergence, we now need to allow for non-zero residuals.

In a regular interior-point method, the residual of the primal-dual optimality conditions is monitored as a termination test. Newton's steps for μ_l are taken until the following criterion is satisfied:

$$\left\| \begin{bmatrix} F_0(u_0^k, u_1^k; \mu_l) \\ F_1(u_0^k, u_1^k; \mu_l) \end{bmatrix} \right\| \leq c_\mu \mu_l. \tag{4.31}$$

Once the tolerance in equation (4.31) is achieved, the barrier parameter is updated according to rules like the following [**81**]:

$$\mu_{l+1} \leftarrow \min\{0.2\mu_l, \mu_l^{1.5}\}. \tag{4.32}$$

Then, the solution to the new barrier problem is initiated. Once the iterates are close to a second-order sufficient solution, one step in the barrier problem already satisfies the optimality conditions, and the barrier parameter is reduced again. This process leads to fast local convergence.

To mimic the strategy of fast local convergence, we need to incorporate additional computations into our algorithm. First, we compute the Newton step by solving the linear system (4.30). This step helps us determine the direction in which we should update our

iterates. Additionally, we calculate the overall residual using equation (4.31), which serves as a termination criterion for the algorithm.

Algorithm 9 presents a potential strategy for the extrapolation step. This algorithm performs the extrapolation step within the decomposition framework. It starts with the regular decomposition step and checks if the termination test $\|F_0(u_0^k, u_1^k)\| \leq c_\mu \mu_l$ is satisfied. If it is, the algorithm decreases μ using equation (4.32) and computes the Newton step using equation (4.30). The new overall iterate is then computed with the fraction-to-the-boundary parameter α_{ftbr}. If the termination test (4.31) holds for the new iterate, it is accepted and the algorithm proceeds to decrease μ again. Otherwise, the algorithm checks the progress of the first-stage problem objectives computed from the extrapolated iterates. If there is progress, the corresponding iterate is accepted, otherwise, it is discarded. The algorithm then repeats the process by going back to the regular decomposition step.

4.7. Numerical Results

In this chapter, we present the numerical results of our proposed two-stage optimization algorithm. We provide a comprehensive evaluation of the algorithm's performance using various scenarios and real-life applications.

First, we illustrate the behavior of the algorithm through a simple example to demonstrate its effectiveness. Next, we analyze the parallel scalability of the algorithm on a set of two-stage Quadratically Constrained Quadratic Programs (QCQPs) to understand its efficiency in utilizing computational resources.

In addition to the experimental results, we also evaluate the algorithm's performance on two real-life applications: the supply-allocation problem and the linear power flow problem. The supply-allocation problem is derived from a distributionally robust two-stage stochastic

Algorithm 9 Extrapolation step

1: Perform an SQP iteration.
2: **if** Termination test $\|F_0(u_0^k, u_1^k)\| \leq c_\mu \mu_l$ holds **then**
3: Decrease μ_l with (4.32).
4: Solve (4.30) to get $(\Delta u_0^k, \Delta u_1^k)$.
5: Compute new overall iterate

$$(\check{u}_0^{k+1}, \check{u}_1^{k+1}) = (u_0^k, u_1^k) + \alpha_{ftbr}(\Delta u_0^k, \Delta u_1^k),$$

where α_{ftbr} is the step size that makes sure that the faction-to-the-boundary rule is satisfied.
6: **if** Termination test (4.31) holds for $(\check{u}_0^{k+1}, \check{u}_1^{k+1})$ **then**
7: Set $(u_0^{k+1}, u_1^{k+1}) \leftarrow (\check{u}_0^{k+1}, \check{u}_1^{k+1})$ and $k \leftarrow k+1$.
8: Go to step 3.
9: **end if**
10: Extract z^k from u_0^k and compute first-stage problem objective $f_0(z^k) + \hat{f}_1(x_1^k; \mu_{l+1})$ and its gradient. This involves a subproblem solve.
11: Extract $\check{z}^{k+1}$ from $\check{u}_0^{k+1}$ and compute first-stage problem objective $f_0(\check{z}^{k+1}) + \hat{f}_1(\check{x}_1^{k+1}; \mu_{l+1})$ and its gradient. This involves a subproblem solve.
12: **if** $f_0(\check{z}^{k+1}) + \hat{f}_1(\check{x}_1^{k+1}; \mu_{l+1}) \leq f_0(z^k) + \hat{f}_1(x_1^k; \mu_{l+1})$ **then**
13: Set $z^{k+1} \leftarrow \check{z}^{k+1}$ and $k \leftarrow k+1$.
14: **else**
15: Discard $\check{z}^{k+1}$ and let z^k unchanged.
16: **end if**
17: **end if**
18: Go to step 1

programming problem [**24**], and we compare the performance of the algorithm using the natural approximation and the log-barrier approximation in different problem settings. For the linear power flow problem, we specifically investigate the impact of the extrapolation step. The results obtained from this real-life application demonstrate the effectiveness of the extrapolation step in improving the overall performance of the algorithm. This evaluation showcases the algorithm's capability to handle complex power flow problems efficiently and accurately.

All computations are performed on a Linux machine equipped with a 2.1GHz Intel Xeon Gold 5128 R CPU and 256 GB of memory. The machine is equipped with 40 cores. The code is compiled using GCC 11.3.0. To solve the second-stage problems and the quadratic

subproblems in the SQP algorithm, we utilize the Ipopt solver with the MA27 linear solver. Parallelization is implemented using OpenMP to leverage the available computational resources efficiently.

Unless stated otherwise, the algorithm starts with an initial smoothing parameter $\mu^0 = 10^{-3}$ and declares convergence when μ reaches $\tau = 10^{-6}$. The smoothing parameter is decreased using a formula $\mu^{k+1} = \max(0.2\mu^k, \tau)$. The parameter τ_{master}, representing the tolerance of the master solver, is set to $0.1\mu^k$ for each iteration k to control the convergence criteria. For all instances, the algorithm is initialized with unit initial primal points for both the master and subproblems. Specifically, the initial primal points are set as $(1 \ \cdots \ 1)$. Additionally, trivial initial multipliers are employed during the algorithm's execution.

4.7.1. Simple Example Illustration

We illustrate the behavior of the two-stage optimization algorithm through a simple example presented in Example 4.3.1. That is

$$\min_{x \in \mathbb{R}^2} \quad \hat{f}(x),$$

where

$$\begin{aligned} \hat{f}(x) = \min_{y \in \mathbb{R}} \quad & y \\ \text{s.t} \quad & y \geq 2x_1, \\ & y \geq x_2, \\ & y \geq -x_1 - x_2. \end{aligned}$$

Table 4.1 summarizes the algorithm's performance for different initial values of the smoothing parameter μ. The table includes the initial μ value, the total number of master

Table 4.1. Summary of the algorithm's performance for different initial values of the smoothing parameter (μ) on Example 4.3.1.

initial μ	iter	#obj eval	#grad eval	avg subiter
5.0×10^{-1}	16	26	26	3.19
1.0×10^{-1}	16	25	22	3.25
2.0×10^{-2}	21	29	22	3.71
4.0×10^{-3}	25	32	22	3.96
8.0×10^{-4}	31	37	22	4.23
1.6×10^{-4}	31	36	19	4.26
3.2×10^{-5}	37	41	20	4.11
1.0×10^{-6}	43	44	16	4.19

solver iterations required for convergence, the number of master problem objective function evaluations, the number of master problem gradient evaluations, and the average number of iterations within each subproblem solve using Ipopt.

The table highlights the impact of the initial μ value on the algorithm's performance. As the initial μ decreases, the number of iterations, objective function evaluations, gradient evaluations, and the average number of iterations within each subproblem solves using Ipopt increase. However, it's important to note that when the initial μ value is too large, such as 5.0×10^{-1}, the number of function evaluations may also increase, and even the iteration numbers can increase.

These findings emphasize the significance of selecting an appropriate initial μ value, as it can significantly affect the convergence behavior of the algorithm. In general, a larger initial μ value leads to faster convergence. However, it's crucial to consider the specific problem and find a balance between convergence speed and computational efficiency when determining the optimal initial μ value.

Additionally, Table 4.2 gives detailed running statistics when using the initial smoothing parameter 1.0e-1. Each row in the table corresponds to an outer iteration of the algorithm, and the columns represent the number of main solver iterations for each specific μ, the

objective function value at the end of an outer iteration, and the average number of Ipopt iterations per SQP iteration for a specific μ. The results show that as the value of μ decreases, the number of iterations decreases as well. This observation suggests that the warm-start option of Ipopt is effective in leveraging previous solutions and accelerating the convergence of subsequent subproblems.

Table 4.2. Summary of results for running Example 4.3.1 using initial smoothing parameter 1.0e-01

μ	objective	iter	avg subiter
1.00e-01	1.9676075787642622e+00	6	5.67
2.00e-02	6.1884278735360931e-01	1	3.00
4.00e-03	1.6883277819541642e-01	2	2.50
8.00e-04	4.2779367948723813e-02	1	2.00
1.60e-04	1.0358404051701563e-02	1	2.00
3.20e-05	2.4321549027190212e-03	2	1.50
6.40e-06	5.5849379902085184e-04	1	1.00
1.28e-06	1.2607932349957993e-04	1	1.00
1.00e-06	1.0021655452956751e-04	1	1.00

4.7.2. Two-stage Quadratically Constrained Quadratic Programs

We consider a randomly generated two-stage QCQP problem in the following form:

$$\min_{x\in\mathbb{R}^{n_0}} \quad \tfrac{1}{2}x^TQ_0x + c_0^Tx + \sum_{i=1}^{N}\hat{f}_i(x_i) \tag{4.33a}$$

$$\text{s.t.} \quad r_{0j} + 0.5x^TQ_{0j}x + c_{0j}^Tx \le 0, \qquad j = 1, ..., m_0, \tag{4.33b}$$

with

$$\hat{f}_i(\hat{x}) = \min_{y_i\in\mathbb{R}^{n_i}} \quad \tfrac{1}{2}y_i^TQ_iy_i + c_i^Ty_i \tag{4.34a}$$

$$\text{s.t.} \quad r_{ij} + 0.5y_i^TQ_{ij}y_i + c_{ij}^Ty_i + b_{ij}^T\hat{x} \le 0, \quad j = 1, ..., m_i. \tag{4.34b}$$

The matrices Q_0 and Q_i, $i = 0, \cdots, N$, are diagonal matrices with non-negative entries. The vector $\hat{x} \in \mathbb{R}^{n_c}$ is a subvector of x that connects the first-stage and second-stage variables, where $\hat{x}$ corresponds to the first n_c entries of x.

To generate test data for the QCQP problems, we follow the procedure described in Algorithm 10. In this set of experiments, we choose $n_0 \in \{50, 100\}$ and $n_i = 10n_0$. The number of constraints for the first stage is $m_0 = 10$, and for the second-stage problems, we consider different numbers of constraints: $m_i \in \{5, 10, 20, 50\}$. The number of subproblems takes values of $N \in \{32, 64, 128, 256, 1024\}$. Additionally, we set the number of connected components as $n_c = 10$. The number of active constraints m_0^A is set to be $0.2m_0$ and m_i^A is set to be $0.2m_i$. The matrix density is set to be 0.05.

The test was conducted using AMPL interfaces, and all subproblem solves was executed in parallel with 32 threads to leverage parallel computing capabilities. The performance of the algorithm on the set of QCQP problems is illustrated in Figure 4.6. Specifically, Figure 4.6a presents the algorithm's performance for $n_0 = 50$, while Figure 4.6b shows the performance for $n_0 = 100$. In both figures, the different lines represent the number of constraints for the subproblems. We observe that the running time exhibits a linear relationship as the number of subproblems increases.

Additionally, we analyze the impact of the number of threads on the running time for a single problem instance. Figure 4.7 illustrates this relationship, indicating the reduction in computational time achieved by solving subproblems in parallel. The relationship between the number of threads and running time exhibits a log-linear pattern. However, it is noteworthy that the reduction in running time from increasing the number of threads from 16 to 32 is not significant. This observation may be attributed to the limited parallelism available

Algorithm 10 QCQP test data generation

Require: The number of first-stage variables n_0; the number of second-stage variables n_i; the number of first-stage constraints m_0; the number of second-stage constraints m_i; the number of subproblems N; the number of connected components n_c. Matrix density d of Q_{0j} and Q_{ij}. A pre-specified optimal solution x^*. y_i^*. Number of active constraints $m_0^A.m_i^A$ for $i \in [N]$.

1: Generate Q_0 as a diagonal matrix with positive entries from uniform$(0.1, 1)$.
2: Generate Q_i similarly for $i \in [N]$.
3: Generate Q_{0j} as diagonal matrices with density d for diagonal entries, nonzero entries from uniform$(0, 1)$.
4: Generate Q_{ij} similarly for $i \in [N]$.
5: Sample c_{0j} entries from the standard normal distribution, setting entries corresponding to zero diagonal entries of Q_{0j} to zero.
6: Sample c_{ij} similarly for $i \in [N]$.
7: Sample b_{ij} entries from standard normal distribution for $i \in [N]$, $j \in [m_i]$.
8: Generate optimal multiplier λ_0^* for (4.33b). Select the first m_0^A of entries as non-zero, sample from uniform$(0.1, 1)$.
9: Calculate r_{0j} as

$$r_{0j} = \begin{cases} -\frac{1}{2}x^*Q_{0j}x^* - c_{0j}^T x^*, & \text{if } j \le m_0^A, \\ -\frac{1}{2}x^*Q_{0j}x^* - c_{0j}^T x^* - \epsilon_{0j}, & \text{otherwise,} \end{cases}$$

where ϵ_{0j} are sampled from uniform$(0.1, 1)$.
10: Calculate r_{ij} as

$$r_{ij} = \begin{cases} -\frac{1}{2}y_i^*Q_{ij}y_i^* - c_{ij}^T y_i^* - b_{ij}^T \hat{x}^*, & \text{if } j \le m_i^A, \\ -\frac{1}{2}y_i^*Q_{ij}y_i^* - c_{ij}^T y_i^* - b_{ij}^T \hat{x}^* - \epsilon_{ij}, & \text{otherwise,} \end{cases}$$

where ϵ_{ij} are sampled from uniform$(0.1, 1)$.
11: Generate λ_i^* for (4.34b). Here we assume that λ_i^* is the same for all i.
12: Calculate $c_0 = -Q_0x^* - \sum_{j=1}^{m_0^A}(Q_{0j}x^* + c_{0j}) \cdot \lambda_{0j}^*$ first, then subtract the first n_c entries of c_0 by $\sum_{i=1}^{n}\sum_{j=1}^{m_i} b_{ij}\lambda_{ij}^*$.
13: Calculate $c_i = -Q_iy_i^* - \sum_{j=1}^{m_i^A}(Q_{ij}y_i^* + c_{ij}) \cdot \lambda_{ij}^*$.
14: **return** Q_0, Q_i, Q_{0j}, Q_{ij}, c_0, c_i, c_{0j}, c_{ij}, b_{ij}, r_{0j}, r_{ij}.

in the master problem solver. Despite this, parallel computing still provides notable benefits, as evidenced by the overall reduction in running time with increasing thread counts.

Table 4.3. Notation for the supply allocation problem

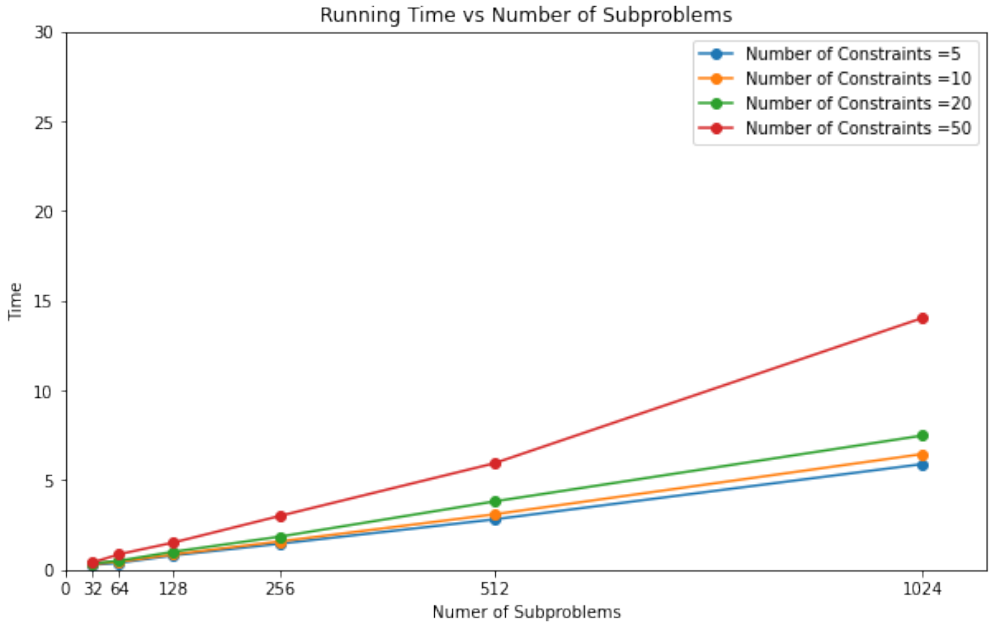
Sets	
$g \in \mathcal{G}$	Set of facilities
$d \in \mathcal{D}$	Set of demand sites
Parameters	
$\ell_g > 0$	Upper limit on supply installed at facility g
$c_{gd} > 0$	Unit cost of satisfying demand at site d from facility g
$\rho > 0$	Unit cost for subcontracted demand
$h > 0$	Unit cost for holding inventory
ξ_{id}	Random demand at site d of realization i
Decision variables	
x_g	Supply allocated to facility g
y_{gd}	Amount supplied by facility g to site d
u_d	Shortfall subcontracted at site d
v_g	Excess supply held at facility g

4.7.3. Supply-allocation problem

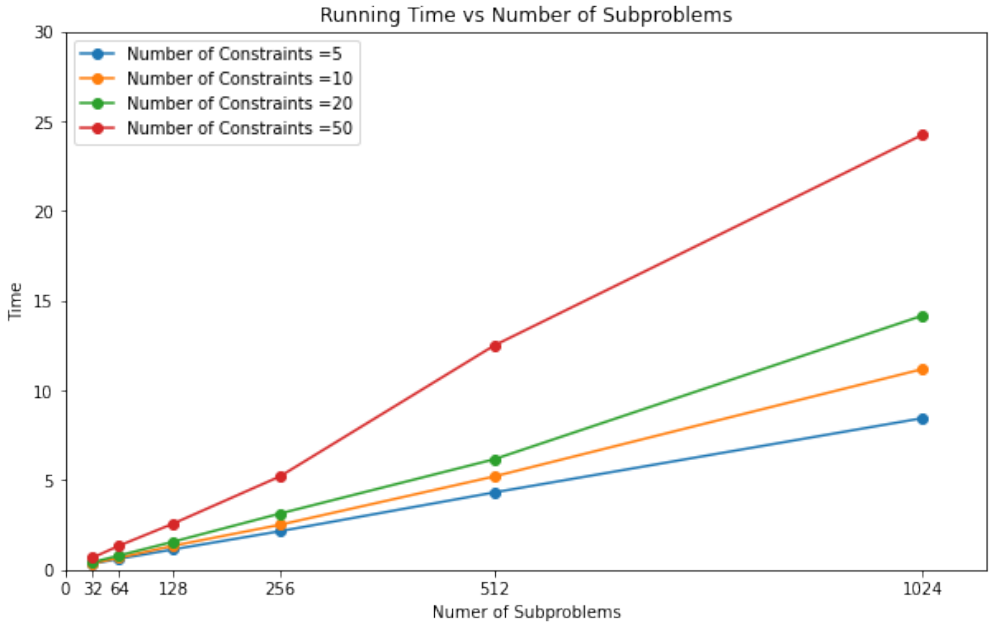
We consider a deterministic version of a two-stage supply-allocation problem introduced in [**24**]. The problem involves allocating supplies to facilities in the first stage and satisfying demand at minimum cost in the second stage. When supply is insufficient, a penalty cost is incurred for each unit of subcontracted demand, and holding costs are incurred for excess supply at facilities. Table 4.3 provides an overview of the notation used in the problem formulation. The two-stage optimization problem is formulated as

$$\min_x \frac{1}{N} \sum_{i=1}^{N} V(x, \xi_i)$$

$$\text{s.t. } 0 \leq x \leq \ell,$$



(a) $n_0 = 50$



(b) $n_0 = 100$

Figure 4.6. Two-stage algorithm performance for QCQP problems. The x-axis represents the number of subproblems. The y-axis represents the running time (in seconds). Each line corresponds to a different number of constraints for the subproblems.

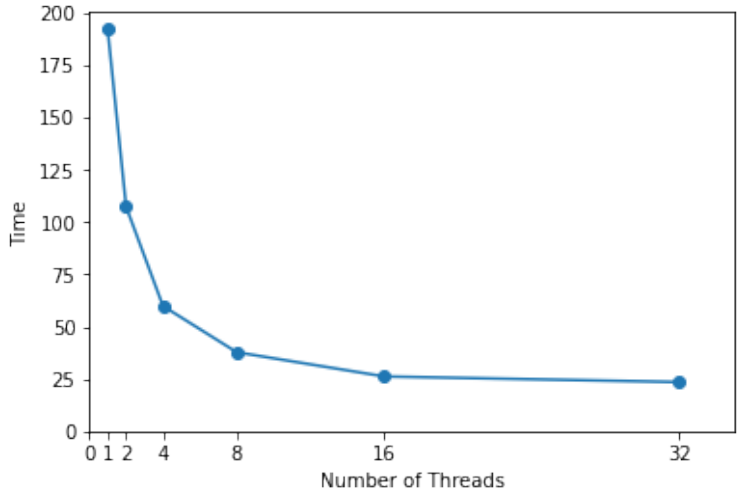


Figure 4.7. Parallel scalability: impact of the number of threads on running time on a two-stage QCQP problem. $n_0 = 100$, $n_i = 1000$, $N = 1024$, $m_0 = 10$, $m_i = 50$. The x-axis represents the number of threads used for parallel computation. The y-axis represents the corresponding running time in seconds.

where

$$V(x, \xi_i) = \min_{y,u,v} \sum_{g\in\mathcal{G}} \sum_{d\in\mathcal{D}} c_{gd} y_{gd} + \sum_{d\in\mathcal{D}} \rho u_d + \sum_{g\in\mathcal{G}} h v_g$$

$$\text{s.t.} \sum_{d\in\mathcal{D}} y_{gd} + v_g = x_g \quad \forall g \in \mathcal{G},$$

$$\sum_{g\in\mathcal{G}} y_{gd} + u_d \geq \xi_{id} \quad \forall d \in \mathcal{D},$$

$$y_{gd}, u_d, v_g \geq 0 \quad \forall g \in \mathcal{G}, d \in \mathcal{D}.$$

We follow a similar setup as described in [**24**], with $|\mathcal{G}| \in \{5, 10, 20\}$, $|\mathcal{D}| \in \{20, 30, 50\}$, $\rho = 10$, and $h = 1$. The unit cost of satisfying demand, c_{gd}, is determined as the Euclidean distance between facility g and demand site d. The locations of both facilities and demand sites are randomly generated within the unit square. The limit on installed capacity, ℓ_g, is uniformly generated from the interval $[100, 200]$. We consider 1000 realizations for the random demands. Each demand is generated from a lognormal distribution with parameters

$\mu = 1$ and $\sigma = 1$. We assume that demands were independent and identically distributed across sites.

Table 4.4 provides a comparison between the natural approximation and the log-barrier approximation in the context of the supply allocation problem. The table includes the number of SQP iterations (iter), the average number of Ipopt iterations per SQP iteration (avg subiter), and the total running time (in seconds) for each combination of facilities and demand sites. We want to note that for the instance with 10 facilities and 50 demand sites, the algorithm using the natural approximation converged to a non-optimal solution.

The results demonstrate notable differences between the two approximations. In terms of the number of iterations, the log-barrier approximation generally requires fewer SQP iterations compared to the natural approximation for the same problem instance. Regarding computational time, the log-barrier approximation also outperforms the natural approximation. The total running time for solving the problem is lower when using the log-barrier approximation compared to the natural approximation for the same problem instance. This indicates that the log-barrier approximation is more computationally efficient and requires less time to obtain the solution. The reduced computational time for the log-barrier approximation can be attributed to its more efficient derivative computation procedure.

4.7.4. Linear power flow problem

We conducted an experiment on a two-stage linear power flow problem, where we optimized the transmission and distribution systems. The first-stage problem focused on the optimization of the transmission system using the IEEE PGLib 500-bus test system, which involved 870 variables and 466 constraints. The second-stage problems involved optimizing the distribution system using the IEEE LVTestCase, with five subproblems each comprising 403

facilities	demand sites	log-barrier approximation			natural approximation		
		iter	avg subiter	time	iter	avg subiter	time
5	20	49	3.79	3.4751	231	4.25	25.6457
5	30	62	4.72	5.7516	253	4.48	31.7653
5	50	38	4.50	3.6483	206	4.33	29.4467
10	20	74	4.69	9.3833	303	5.02	81.6899
10	30	69	5.41	10.8582	211	3.61	60.6383
10	50	54	5.27	9.9066	272	3.96	97.4558
20	20	133	4.94	38.0356	412	4.78	402.2237
20	30	85	6.21	29.4327	407	5.01	477.3704
20	50	117	5.24	46.0824	335	5.66	526.6822

Table 4.4. Summary of the results for the supply allocation problem, with 1000 subproblems. facilities represent the number of facilities in the problem instance. demand sites represent the number of demand sites in the problem instance. iter denotes the number of SQP iterations performed by the algorithm. avg subiter denotes the average number of Ipopt iterations for solving all the second-stage problems per SQP iteration. The time represents the total running time (in seconds) the algorithm takes to solve the problem. For the instance with 10 facilities and 50 demand sites, the algorithm using the natural approximation failed to converge to an optimal solution.

variables and 394 constraints. For detailed formulations, please refer to [**61, 77, 76**]. The test instance is running directly from PowerModelsITDs.jl [**61**].

In Table 4.5, we present a comparison of the algorithm's performance with and without the extrapolation step. The table provides detailed running statistics for each smoothing parameter value during the outer iteration, with the algorithm terminating when μ reaches 1e-5. The results reveal several observations. Firstly, the algorithm with the extrapolation step demonstrates a faster convergence rate at the master solver level, requiring fewer iterations to converge after the first outer iteration. Furthermore, at the subproblem level, the use of extrapolation steps leads to a reduced average number of iterations per subproblem solved with Ipopt.

	w/o extrapolation		w/ extrapolation	
μ	iter	avg subiter	iter	avg subiter
1e-1	65	6.09	65	6.09
2e-2	9	3.84	5	2.60
4e-3	10	3.90	4	2.25
8e-4	11	3.76	3	2.00
1.6e-4	11	3.38	3	1.67
3.2e-5	11	3.55	2	1.50
1e-5	7	3.74	2	1.50

Table 4.5. Comparison of results on the linear power flow problem with and without the extrapolation step. Each row represents an outer iteration with a specific μ value. The "iter" column denotes the number of SQP iterations performed, while the "avg subiter" column indicates the average number of Ipopt iterations incurred per subproblem solve per SQP iteration.

4.8. Concluding Remarks and Future Directions

In this chapter, we present a novel advancement in the field of continuous nonlinear two-stage optimization problems. Our approach builds upon a new approximation scheme based on the smoothing technique proposed by [**77**]. By leveraging this innovative technique, we achieve faster convergence speeds and significantly improve computational efficiency.

We have developed a new C++ software package that enables the practical application of the algorithm. This software allows for the practical application of the algorithm and provides valuable insights into its performance through extensive numerical experiments.

Looking ahead, our software is poised for further development and expansion. We aim to extend its capabilities to tackle even larger-scale two-stage nonlinear optimization problems, addressing more challenging real-world applications, like nonlinear ACOPF. Additionally, we envision potential advancements in handling general nonlinear bilevel optimization problems, broadening the scope of its applicability and impact.

References

[1] Aiyoshi, E., and Shimizu, K. Hierarchical decentralized systems and its new solution by a barrier method. *IEEE Transactions on Systems, Man and Cybernetics*, 6 (1981), 444–449.

[2] Alizadeh, F., and Goldfarb, D. Second-order cone programming. *Mathematical Programming 95*, 1 (2003), 3–51.

[3] ApS, M. *The MOSEK optimization toolbox for MATLAB manual. Version 9.1*, 2019.

[4] Barbarosoğlu, G., and Arda, Y. A two-stage stochastic programming framework for transportation planning in disaster response. *Journal of the Operational Research Society 55* (2004), 43–53.

[5] Ben-Tal, A., and Nemirovski, A. On polyhedral approximations of the second-order cone. *Mathematics of Operations Research 26*, 2 (2001), 193–205.

[6] Benders, J. F. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math. 4*, 1 (1962), 238–252.

[7] Best, M. J. An algorithm for the solution of the parametric quadratic programming problem. In *Applied mathematics and parallel computing*. Springer, 1996, pp. 57–76.

[8] Biggs, M. Constrained minimization using recursive equality quadratic programming. *Numerical Methods for Nonlinear Optimization* (1972), 411–428.

[9] Birge, J. R., and Louveaux, F. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[10] Bitar, G., Vestad, V. N., Lekkas, A. M., and Breivik, M. Warm-started optimized trajectory planning for asvs. *IFAC-PapersOnLine 52*, 21 (2019), 308–314.

[11] Boggs, P. T., and Tolle, J. W. Sequential quadratic programming. *Acta numerica 4* (1995), 1–51.

[12] Bongartz, I., Conn, A. R., Gould, N., and Toint, P. L. Cute: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software 21*, 1 (1995), 123–160.

[13] Braun, R. D. *Collaborative optimization: an architecture for large-scale distributed design*. PhD thesis, Stanford University, 1996.

[14] Brown, N. F., and Olds, J. R. Evaluation of multidisciplinary optimization techniques applied to a reusable launch vehicle. *Journal of Spacecraft and Rockets 43*, 6 (2006), 1289–1300.

[15] Burke, J. V. A robust trust region method for constrained nonlinear programming problems. *SIAM Journal on Optimization 2*, 2 (1992), 325–347.

[16] Byrd, R. H., Nocedal, J., and Waltz, R. A. Steering exact penalty methods for nonlinear programming. *Optimization Methods and Software 23*, 2 (2008), 197–213.

[17] Coey, C., Lubin, M., and Vielma, J. P. Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation 12*, 2 (2020), 249–293.

[18] Conn, A. R., Gould, N. I., and Toint, P. L. *Trust region methods*. SIAM, 2000.

[19] DeMiguel, A.-V., and Murray, W. An analysis of collaborative optimization methods. In *8th symposium on multidisciplinary analysis and optimization* (2000), p. 4720.

[20] DeMiguel, V., and Murray, W. A local convergence analysis of bilevel decomposition algorithms. *Optimization and Engineering 7* (2006), 99–133.

[21] Diehl, M., Jarre, F., and Vogelbusch, C. H. Loss of superlinear convergence for an sqp-type method with conic constraints. *SIAM Journal on Optimization 16*, 4 (2006), 1201–1210.

[22] Dogan, M. S., Lund, J. R., and Medellin-Azuara, J. Hybrid linear and nonlinear programming model for hydropower reservoir optimization. *Journal of Water Resources Planning and Management 147*, 3 (2021), 06021001.

[23] Drewes, S., and Ulbrich, S. Subgradient based outer approximation for mixed integer second order cone programming. In *Mixed integer nonlinear programming*. Springer, 2012, pp. 41–59.

[24] Duque, D., Mehrotra, S., and Morton, D. P. Distributionally robust two-stage stochastic programming. *SIAM Journal on Optimization 32*, 3 (2022), 1499–1522.

[25] Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation 6*, 4 (2014), 327–363.

[26] Fletcher, R. Second order corrections for non-differentiable optimization. *Numerical Analysis* (1982), 85–114.

[27] Fletcher, R. *Practical methods of optimization*. John Wiley & Sons, 2013.

[28] Fletcher, R., Gould, N. I., Leyffer, S., Toint, P. L., and Wächter, A. Global convergence of a trust-region sqp-filter algorithm for general nonlinear programming. *SIAM Journal on Optimization 13*, 3 (2002), 635–659.

[29] Fletcher, R., Leyffer, S., and Toint, P. L. On the global convergence of a filter-sqp algorithm. *SIAM Journal on Optimization 13*, 1 (2002), 44–59.

[30] Fourer, R., Gay, D. M., and Kernighan, B. W. *AMPL: A mathematical programming language*. AT & T Bell Laboratories Murray Hill, NJ, 1987.

[31] Friberg, H. A. Cblib 2014: a benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation 8* (2016), 191–214.

[32] Geoffrion, A. M. Generalized benders decomposition. *Journal of Optimization Theory and Applications 10* (1972), 237–260.

[33] Gill, P. E., Murray, W., and Saunders, M. A. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review 47*, 1 (2005), 99–131.

[34] Gill, P. E., Murray, W., and Wright, M. H. *Practical optimization*. SIAM, 2019.

[35] Goldberg, N., and Leyffer, S. An active-set method for second-order conic-constrained quadratic programming. *SIAM Journal on Optimization 25*, 3 (2015), 1455–1477.

[36] Gondzio, J. Warm start of the primal-dual method applied in the cutting-plane scheme. *Mathematical Programming 83*, 1-3 (1998), 125–143.

[37] Gondzio, J., and Grothey, A. A new unblocking technique to warmstart interior point methods based on sensitivity analysis. *SIAM Journal on Optimization 19*, 3 (2008), 1184–1210.

[38] Goswami, N., Mondal, S. K., and Paruya, S. A comparative study of dual active-set and primal-dual interior-point method. *IFAC Proceedings Volumes 45*, 15 (2012).

[39] Gould, N., and Toint, P. L. A quadratic programming page. `https://www.numerical.rl.ac.uk/people/nimg/qp/qp.html`.

[40] Gould, N. I., Orban, D., Sartenaer, A., and Toint, P. L. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. *SIAM Journal on Optimization 11*, 4 (2001), 974–1002.

[41] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2022.

[42] Haftka, R. T., and Watson, L. T. Multidisciplinary design optimization with quasiseparable subsystems. *Optimization and Engineering 6* (2005), 9–20.

[43] Han, S.-P. Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Mathematical Programming 11*, 1 (1976), 263–282.

[44] Hayashi, S., Okuno, T., and Ito, Y. Simplex-type algorithm for second-order cone programmes via semi-infinite programming reformulation. *Optimization Methods and Software 31*, 6 (2016), 1272–1297.

[45] Hei, L., Nocedal, J., and Waltz, R. A. A numerical study of active-set and interior-point methods for bound constrained optimization. In *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the Third International Conference on High Performance Scientific Computing, March 6–10, 2006, Hanoi, Vietnam* (2008), Springer, pp. 273–292.

[46] Hijazi, H., Coffrin, C., and Van Hentenryck, P. Polynomial sdp cuts for optimal power flow. In *2016 Power Systems Computation Conference (PSCC)* (2016), IEEE, pp. 1–7.

[47] ILOG, I. *User's Manual for CPLEX V12.9.0.*, 2019.

[48] Ishizuka, Y., and Aiyoshi, E. Double penalty method for bilevel optimization problems. *Annals of Operations Research 34*, 1 (1992), 73–88.

[49] Kato, H., and Fukushima, M. An sqp-type algorithm for nonlinear second-order cone programs. *Optimization Letters 1*, 2 (2007), 129–144.

[50] Kim, Y., and Anitescu, M. A real-time optimization with warm-start of multiperiod ac optimal power flows. *Electric Power Systems Research 189* (2020), 106721.

[51] Lau, M. S. A comparison of interior point and active set methods for fpga implementation of model predictive control. In *2009 European Control Conference (ECC)* (2009), IEEE, pp. 156–161.

[52] Liu, C., Fan, Y., and Ordóñez, F. A two-stage stochastic programming model for transportation network protection. *Computers & Operations Research 36*, 5 (2009), 1582–1590.

[53] Lubin, M., Dowson, O., Garcia, J. D., Huchette, J., Legat, B., and Vielma, J. P. Jump 1.0: recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation* (2023), 1–9.

[54] Luenberger, D. G., and Ye, Y. *Linear and nonlinear programming*, vol. 2. Springer, 1984.

[55] Maratos, N. *Exact penalty function algorithms for finite dimensional and control optimization problems*. PhD thesis, Imperial College London, 1978.

[56] Marley, J. F., Molzahn, D. K., and Hiskens, I. A. Solving multiperiod opf problems using an ac-qp algorithm initialized with an socp relaxation. *IEEE Transactions on Power Systems 32*, 5 (2016), 3538–3548.

[57] Molzahn, D. K., and Hiskens, I. A. A survey of relaxations and approximations of the power flow equations. *Foundations and Trends in Electric Energy Systems 4*, 1-2 (2019), 1–221.

[58] Nayakakuppam, M., Overton, M., and Schemita, S. *SDPpack user's guide-version 0.9 Beta*, 1997.

[59] Nocedal, J., and Wright, S. *Numerical optimization*. Springer Science & Business Media, 2006.

[60] Okuno, T., Yasuda, K., and Hayashi, S. Sl1qp based algorithm with trust region technique for solving nonlinear second-order cone programming problems. *Interdisciplinary Information Sciences 21*, 2 (2015), 97–107.

[61] Ospina, J., Fobes, D. M., Bent, R., and Wächter, A. Modeling and rapid prototyping of integrated transmission-distribution opf formulations with powermodelsitd. jl. *IEEE Transactions on Power Systems* (2023).

[62] Paiva, L. T., and Fontes, F. Adaptive time-mesh refinement in optimal control problems with state constraints. *Discrete and Continuous Dynamical Systems 35*, 9 (2015), 4553–4572.

[63] Potra, F. A., and Wright, S. J. Interior-point methods. *Journal of Computational and Applied Mathematics 124*, 1-2 (2000), 281–302.

[64] Powell, M. J. The convergence of variable metric methods for nonlinearly constrained optimization calculations. In *Nonlinear programming 3*. Elsevier, 1978, pp. 27–63.

[65] Prékopa, A. *Stochastic programming*, vol. 324. Springer Science & Business Media, 2013.

[66] Raghunathan, A. U., Gopal, V., Subramanian, D., Biegler, L. T., and Samad, T. Dynamic optimization strategies for three-dimensional conflict resolution of multiple aircraft. *Journal of guidance, control, and dynamics 27*, 4 (2004), 586–594.

[67] Robinson, S. M. Perturbed kuhn-tucker points and rates of convergence for a class of nonlinear-programming algorithms. *Mathematical Programming 7* (1974), 1–16.

[68] Schanen, M., Gilbert, F., Petra, C. G., and Anitescu, M. Toward multiperiod ac-based contingency constrained optimal power flow at large scale. In *2018 Power Systems Computation Conference (PSCC)* (2018), IEEE, pp. 1–7.

[69] Schittkowski, K. Nlpqlp: A fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search–user's guide. Tech. rep., Department of Computer Science, University of Bayreuth, 2006.

[70] Schork, L. A parametric active set method for general quadratic programming. Master's thesis, Heidelberg University, Germany, 2015.

[71] Shimizu, K., and Aiyoshi, E. A new computational method for stackelberg and min-max problems by use of a penalty method. *IEEE Transactions on Automatic Control 26*, 2 (1981), 460–466.

[72] Sobieszczanski-Sobieski, J., Altus, T. D., Phillips, M., and Sandusky, R. Bilevel integrated system synthesis for concurrent and distributed processing. *AIAA journal 41*, 10 (2003), 1996–2003.

[73] Tammer, K. The application of parametric optimization and imbedding to the foundation and realization of a generalized primal decomposition approach. *Mathematical research 35* (1987), 376–386.

[74] Tsaknakis, I., Khanduri, P., and Hong, M. An implicit gradient method for constrained bilevel problems using barrier approximation. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023), IEEE, pp. 1–5.

[75] Tseng, P. Second-order cone programming relaxation of sensor network localization. *SIAM Journal on Optimization 18*, 1 (2007), 156–185.

[76] Tu, S. *Two-Stage Decomposition Algorithms and Their Application to Optimal Power Flow Problems*. PhD thesis, Northwestern University, 2021.

[77] Tu, S., Wächter, A., and Wei, E. A two-stage decomposition approach for ac optimal power flow. *IEEE Transactions on Power Systems 36*, 1 (2020), 303–312.

[78] Van Slyke, R. M., and Wets, R. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics 17*, 4 (1969), 638–663.

[79] Vanderbei, R. J. Cute models. https://vanderbei.princeton.edu/ampl/nlmodels/cute/.

[80] Vanderbei, R. J., and Yurttan, H. Using loqo to solve second-order cone programming problems. Tech. rep., Priceton University, 1998.

[81] Wächter, A., and Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming 106*, 1 (2006), 25–57.

[82] Waltz, R. A., and Nocedal, J. Knitro 2.0 user's manual. *Ziena Optimization, Inc.[en ligne] disponible sur http://www. ziena. com (September, 2010) 7* (2004), 33–34.

[83] Wilson, R. B. *A simplicial algorithm for concave programming*. PhD thesis, Harvard University, 1963.

[84] Yildirim, E. A., and Wright, S. J. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization 12*, 3 (2002), 782–810.

[85] Yuan, Y.-X. On the superlinear convergence of a trust region algorithm for nonsmooth optimization. *Mathematical Programming 31*, 3 (1985), 269–285.

[86] Zhadan, V. The variant of primal simplex-type method for linear second-order cone programming. In *Optimization and Applications: 12th International Conference, OPTIMA 2021, Petrovac, Montenegro, September 27–October 1, 2021, Proceedings* (2021), Springer, pp. 64–75.

[87] Zhang, X., Liu, Z., and Liu, S. A trust region sqp-filter method for nonlinear second-order cone programming. *Computers & Mathematics with Applications 63*, 12 (2012), 1569–1576.

APPENDIX A

Generation of random instances

The test instances used for the numerical experiments have the form

$$\begin{aligned}
&\min_{x\in\mathbb{R}^n} \sum_{j=1}^{p} c_j^T x_j + c_{p+1}^T x_{p+1} \\
&\text{s.t. } \sum_{j=1}^{p} A_j x_j + A_{p+1} x_{p+1} = b, \\
&\qquad x_j \in \mathcal{K}_j, \quad j \in [p], \\
&\qquad x_{j0} \leq 1000, \quad 0 \leq x_{p+1} \leq 1000,
\end{aligned}$$

where $(c_1, \cdots c_p, c_{p+1}) \in \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_{p+1}}$ is the partition of the objective vector c, and $(A_1, \cdots, A_{p+1})$ is the partition of the column vectors of constraint matrix $A \in \mathbb{R}^{m\times n}$. The subvector x_{p+1} includes all optimization variables that are not in any of the cones.

In contrast to the original SOCP (2.1), this formulation includes linear equality constraints, but it is straightforward to generalize Algorithm 3 and its convergence proofs for this setting. The formulation above also includes large upper bounds on all variables so that the set X defined in (2.9) is bounded. However, the upper bounds were chosen so large that they are not active at the optimal solution.

Algorithm 11 describes how the data for these instances are generated. The algorithm generates an optimal primal-dual solution in a way so that, at the solution, K_0 cones are extremal-active, for K_I cones the optimal solution lies in the interior of the cone, and for

Algorithm 11 Random Instance Generation

Require: n (number of variables), m (number of linear constraints), p (total number of cones), K_0 (number of extremal-active cones), K_I (number of inactive cones), K_B (number of cones active at the boundary, excluding extremal-active cones), d (density of nonzeros in constraint matrix). Conditions: $n, m, p, K_0, K_I, K_B \in \mathbb{Z}_+$, $d \in (0, 1]$, $p = K_0 + K_I + K_B$, $n > 2p$, $n > m + K_B + \lfloor (m + K_B)/2 \rfloor$.

1: Randomly choose positive integers $n_1, \ldots, n_{K_0}$ so that $n_j \geq 2$ and $\sum_{j=1}^{K_0} n_j = \lfloor (m + K_B)/2 \rfloor$.
2: Randomly choose positive integers $n_{K_0+1}, \ldots, n_p$ so that $n_j \geq 2$ and $\sum_{j=K_0+1}^{p} n_j = m + K_B$.
3: **for** $j = 1, \ldots, K_0$ **do** ▷ Generate $x_j^* = 0$, $z_j^* \in \text{int}(\mathcal{K}_j)$
4: Set $x_j^* \leftarrow 0$.
5: Sample $\bar{z}_j \sim (-10, 10)^{n_j - 1}$, $z_{j0}^* \sim (1, 5)$ and $\epsilon_j \sim (0, 1)$. Set $\bar{z}_j^* \leftarrow (2 + \epsilon) \frac{z_{j0}^*}{\|\bar{z}_j\|} \bar{z}_j$.
6: **end for**
7: **for** $j = K_0 + 1, \ldots, K_0 + K_I$ **do** ▷ Generate $z_j^* = 0$, $x_j^* \in \text{int}(\mathcal{K}_j)$
8: Set $z_j^* \leftarrow 0$.
9: Sample $\bar{x}_j \sim (-10, 10)^{n_j - 1}$, $x_{j0}^* \sim (1, 5)$ and $\epsilon_j \sim (0, 1)$. Set $\bar{x}_j^* \leftarrow (2 + \epsilon) \frac{x_{j0}^*}{\|\bar{x}_j\|} \bar{x}_j$.
10: **end for**
11: **for** $j = K_0 + K_I + 1, \ldots, p$ **do** ▷ Generate $z_j^*, x_j^* \in \text{bd}(\mathcal{K}_j)$
12: Sample $\bar{x}_j \sim (-10, 10)^{n_j - 1}$ and $x_{j0}^* \sim (1, 5)$. Set $\bar{x}_j^* \leftarrow \frac{x_{j0}^*}{\|\bar{x}_j\|} \bar{x}_j$.
13: Sample $\beta \sim (1, 5)$. Set $\bar{z}_j^* \leftarrow -\beta \bar{x}_j^*$ and $\bar{z}_{j0}^* \leftarrow \beta x_{j0}^*$
14: **end for**
15: Set $n_{\text{fix}} \leftarrow n - m - K_B - \sum_{j=1}^{K_0} n_j$ and $n_{\text{free}} \leftarrow n - n_{\text{fix}} - \sum_{j=1}^{p} n_j$.
16: **for** $j = 1, \ldots, n_{\text{fix}}$ **do**
17: Set $[x_{p+1}^*]_j \leftarrow 0$ and sample $[z_{p+1}^*]_j \sim (1, 5)$.
18: **end for**
19: **for** $j = n_{\text{fix}} + 1, \ldots, n_{\text{free}}$ **do**
20: Set $[z_{p+1}^*]_j \leftarrow 0$ and sample $[x_{p+1}^*]_j \sim (1, 5)$.
21: **end for**
22: Sample linear independent rows $A_i \sim (-5, 5)^n$ with density d, for $i = 1, \cdots, m$.
23: Call `pcond` to calculate primal condition number c_p and call `dcond` to calculate dual condition number c_d.
24: **if** $c_p > 10^5$ or $c_d > 10^5$ **then** go to Step 2 and redo the process.
25: Sample $\lambda^* \sim (1, 10)^m$. Set $b \leftarrow Ax^*$ and $c \leftarrow -A^T \lambda^* + z^*$.
26: **return** $A, b, c, x^*, \lambda^*, z^*$.

K_B the optimal solution is at the boundary but not the extreme point of the cone. In our experiments, there is an equal number of K of each type. The number of variables that are active at the lower bound, n_{fix}, is chosen in a way so that the optimal solution

is non-degenerate. The linear constraints reduce the number of degrees of freedom by m, each extremal active cone by n_j, and each otherwise active cone by 1. Step 15 calculate the number of constraints that are active at zero so that the total number of degrees of freedom fixed by constraints equals n. Lastly, in Step 23, we use `pcond` and `dcond` functions provided in SDPPack [**58**] to double-check if a generated instance is also numerically non-degenerate. These functions return the primal and dual conditions numbers, c_p and c_d, respectively, and we discard an instance if either number is above a threshold. Only about 1% of instances created were excluded in this manner.

APPENDIX B

CUTE Result

nVar	Number of variables
nConstr	Number of constraints
iter	Number of SQP iteration
QP_iter	Total number of QP iteration
objective	Objective value upon return
#obj	Number of objective function evaluations
#grad	Number of gradient function evaluations
cpu_time	Total CPU time in seconds
exitflag	0 : Optimal -2 : Exceed maximum number of iterations -3 : Predicted reduction is non-negative -4 : Trust-region radius is too small -8 : Penalty parameter is too large -21: QP solver internal error -22: QP solver reports the problem is infeasible -23: QP solver reports the problem is unbounded -24: QP solver exceeds the maximum number of iterations -30: Unknown QP error -32: QP error: Ipopt has restoration phase failed -33: QP error: Ipopt has error in step computation

B.1. With QORE as QP solver

Table B.1. Table of results on the CUTE test set for RestartSQP using QORE as QP subsolver

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
3pk	30	0	6	40	0	1.720118e+00	7	7	0.0017
aircrfta	5	5	2	13	0	0.000000e+00	3	3	0.0005
aircrftb	8	3	20	69	0	1.208028e-23	22	12	0.0015
airport	84	42	12	357	0	4.795270e+04	13	13	0.0263
aljazzaf	3	1	20	46	0	7.500499e+01	24	18	0.0014
allinit	4	3	13	37	0	1.670596e+01	14	9	0.0011
allinitc	4	4	16	58	0	3.048876e+01	18	16	0.0014
allinitu	4	0	9	15	0	5.744384e+00	10	7	0.0006
alsotame	2	3	4	16	0	8.208499e-02	5	5	0.0005
arglina	100	0	1	101	0	1.000000e+02	2	2	0.0131
arglinb	10	0	1	14	0	4.634146e+00	2	2	0.0003
arglinc	8	0	1	10	0	6.135135e+00	2	2	0.0003
argtrig	100	100	3	206	0	0.000000e+00	4	4	0.0503
artif	5000	5000	22	22758	-8	0.000000e+00	36	12	19.1410
arwhead	5000	0	6	5006	0	-2.664535e-15	7	7	1.6963
aug2d	20192	9996	4	84915	0	1.687411e+06	5	5	160.7029
aug2dc	20200	10194	4	85588	0	1.818392e+06	5	5	147.8718
aug2dcqp	20200	10194	0	22754	-21	1.010000e+04	1	1	30.5820
aug2dqp	20192	10194	0	27291	-23	9.900000e+03	1	1	36.3020
aug3d	3873	1000	1	6485	0	5.540677e+02	2	2	2.1362
aug3dc	3873	1000	1	5874	0	7.712624e+02	2	2	1.6051
aug3dcqp	3873	1000	1	5664	0	9.933621e+02	2	2	1.3578
aug3dqp	3873	1000	1	4374	0	6.752376e+02	2	2	1.0953
avgasa	8	10	1	13	0	-4.412171e+00	2	2	0.0003
avgasb	8	10	1	12	0	-4.483219e+00	2	2	0.0003
avion2	49	15	43	232	0	9.468012e+07	48	23	0.0106

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
bard	3	0	7	10	0	8.214877e-03	8	8	0.0005
batch	46	73	7	479	0	2.591803e+05	8	8	0.0237
bdexp	5000	0	10	5011	0	1.963835e-03	11	11	0.6248
bdqrtic	1000	0	9	1009	0	3.983817e+03	10	10	0.0995
beale	2	0	292	431	-3	4.522242e-01	293	206	0.0096
bigbank	2230	1112	20	7059	0	-4.205696e+06	21	21	2.0189
biggs3	6	3	10	29	0	3.115215e-14	12	9	0.0009
biggs5	6	1	38	100	0	2.835358e-02	39	26	0.0037
biggs6	6	0	57	153	0	1.940162e-11	58	34	0.0032
biggsb1	1000	999	1	3993	0	1.499999e-02	2	2	0.4646
biggsc4	4	7	1	28	0	-2.450000e+01	2	2	0.0005
blockqp1	2005	1001	1	1303	0	-9.964999e+02	2	2	1.0076
blockqp2	2005	1001	1	1074	0	-9.961011e+02	2	2	1.0540
blockqp3	2005	1001	1	1990	0	-4.974999e+02	2	2	1.5286
blockqp4	2005	1001	1	1081	0	-4.980982e+02	2	2	1.3056
blockqp5	2005	1001	1	1990	0	-4.974999e+02	2	2	1.3660
bloweya	2002	1002	0	100000	-24	-7.343399e-04	1	1	28.5870
bloweyb	2002	1002	0	838	-21	-6.010044e-04	1	1	0.2646
bloweyc	2002	1002	0	100000	-24	-6.010044e-04	1	1	30.8620
booth	2	2	1	5	0	0.000000e+00	2	2	0.0002
box2	3	1	8	18	0	1.656702e-20	10	6	0.0006
box3	3	0	7	10	0	1.901209e-13	8	7	0.0005
bqp1var	1	1	1	4	0	0.000000e+00	2	2	0.0002
bqpgabim	50	4	1	57	0	-3.790343e-05	2	2	0.0012
bqpgasim	50	0	1	49	0	-5.519814e-05	2	2	0.0008
bratu1d	1001	0	8	1497	0	-8.518927e+00	9	9	0.0864
bratu2d	4900	4900	2	61583	0	0.000000e+00	3	3	114.9517
bratu2dt	4900	4900	5	55081	0	0.000000e+00	6	6	97.2834
bratu3d	3375	3375	3	25675	0	0.000000e+00	4	4	75.1642
britgas	450	360	1	108225	-24	5.822179e+00	2	2	7.2676

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
brkmcc	2	0	2	4	0	1.690426e-01	3	3	0.0003
brownal	10	0	6	24	0	7.976678e-11	7	7	0.0007
brownbs	2	0	47	51	0	0.000000e+00	48	35	0.0014
brownden	4	0	8	12	0	8.582220e+04	9	9	0.0011
broydn3d	10000	10000	4	20002	0	0.000000e+00	5	5	27.5723
broydn7d	1000	0	10	4122	-3	3.1000137+102	11	7	0.3110
broydnbd	5000	5000	5	12672	0	0.000000e+00	6	6	11.4956
brybnd	5000	0	7	5007	0	1.294855e-13	8	8	1.4892
bt1	2	1	8	33	0	-9.999999e-01	11	7	0.0009
bt10	2	2	6	11	0	-1.000000e+00	7	7	0.0005
bt11	5	3	7	21	0	8.248917e-01	8	8	0.0007
bt12	5	3	11	53	0	6.188118e+00	19	8	0.0012
bt13	5	2	15	46	0	0.000000e+00	18	15	0.0010
bt2	3	1	11	15	0	3.256821e-02	12	12	0.0008
bt3	5	3	2	12	0	4.093023e+00	3	3	0.0004
bt4	3	2	3	23	-3	-1.860893e+01	7	1	0.0006
bt5	3	2	47	146	0	9.617151e+02	92	23	0.0032
bt6	5	2	9	22	0	2.770447e-01	12	9	0.0007
bt7	5	3	18	91	0	4.039999e+02	27	12	0.0021
bt8	5	2	14	51	0	1.000006e+00	23	10	0.0012
bt9	4	2	12	55	0	-1.000000e+00	18	10	0.0009
byrdsphr	3	2	85	212	0	-4.683300e+00	170	44	0.0054
camel6	2	2	11	27	0	-2.154638e-01	12	7	0.0006
cantilvr	5	1	61	134	-4	2.914991e-01	123	5	0.0021
catena	32	11	21	513	0	-2.307774e+04	32	12	0.0094
catenary	496	166	15	2043	-21	-1.667768e+06	23	10	0.1418
cb2	3	3	6	17	0	1.952222e+00	8	6	0.0006
cb3	3	3	8	25	0	1.999999e+00	11	7	0.0007
cbratu2d	882	882	1	8382	0	0.000000e+00	2	2	2.6935
cbratu3d	1024	1024	1	3789	0	0.000000e+00	2	2	1.5554

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
chaconn1	3	3	4	10	0	1.952224e+00	5	5	0.0004
chaconn2	3	3	4	9	0	1.999999e+00	5	5	0.0004
chainwoo	1000	0	75	6722	0	7.077184e+01	76	50	0.3375
chandheq	100	100	9	209	0	0.000000e+00	10	10	0.1175
chebyqad	50	0	64	798	0	5.024788e-03	65	31	0.5427
chemrctb	1000	1000	344	19744	-4	0.000000e+00	685	164	6.4453
chenhark	1000	0	1	720	0	-2.000000e+00	2	2	0.0358
chnrosnb	50	0	55	156	0	1.297084e-14	56	42	0.0064
cliff	2	0	27	29	0	1.997866e-01	28	28	0.0011
clnlbeam	1499	1000	0	102843	-24	3.496824e+02	1	1	18.4583
clplatea	4970	0	6	4978	0	-1.259209e-02	7	6	1.9062
clplateb	4970	0	11	5123	0	-6.988222e+00	12	10	1.8305
clplatec	4970	0	1	4971	0	-5.020724e-03	2	2	1.5752
cluster	2	2	12	38	0	0.000000e+00	17	9	0.0009
concon	15	11	8	73	0	-6.230795e+03	12	9	0.0016
congigmz	3	5	5	27	0	2.800000e+01	6	6	0.0008
coolhans	9	9	14	133	0	0.000000e+00	19	11	0.0034
core1	65	115	51	1808	-8	6.059484e+01	75	30	0.0945
core2	157	134	24	1846	-8	7.424797e+01	37	16	0.1060
coshfun	61	20	466	3191	0	-7.732663e-01	932	228	0.2882
cosine	10000	0	5	10007	0	-9.999000e+03	6	6	3.1551
cragglvy	5000	0	24	5024	0	1.688215e+03	25	23	0.7292
cresc100	6	200	26	3769	-3	6.791142e+01	37	5	0.3023
cresc132	6	2654	33	59017	-3	6.783529e+01	43	5	25.5266
cresc4	6	8	450	1161	0	8.718975e-01	895	224	0.0405
cresc50	6	100	3000	7987	-2	5.981437e-01	5996	1497	0.8386
csfi1	5	4	68	262	0	-4.907520e+01	121	36	0.0058
csfi2	5	4	67	229	0	5.501760e+01	127	36	0.0060
cube	2	0	18	26	-3	1.421377e+00	19	12	0.0011
curly10	10000	0	13	10447	0	-9.968383e+05	14	9	7.3853

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
curly20	10000	0	15	13819	0	-9.968383e+05	16	10	17.1323
curly30	10000	0	17	14845	0	-9.968383e+05	18	11	26.5784
cvxbqp1	10000	0	1	1	0	2.250225e+06	2	2	0.0187
cvxqp1	1000	500	1	5140	0	1.087511e+06	2	2	0.9898
cvxqp2	10000	2500	1	16726	0	8.184245e+07	2	2	21.6840
cvxqp3	10000	7500	0	107529	-24	5.625562e+07	1	1	172.1075
deconvb	51	0	38	629	0	3.352686e-03	39	21	0.0215
deconvc	51	1	59	1224	0	6.241270e-09	119	24	0.0454
deconvu	51	0	958	14620	-3	7.751394e-09	959	475	0.5537
degenlpa	20	14	1	112	0	3.060349e+00	2	2	0.0042
degenlpb	20	15	1	124	0	-3.073124e+01	2	2	0.0050
demymalo	3	3	6	17	0	-3.000000e+00	8	6	0.0010
denschna	2	0	5	7	0	2.213909e-12	6	6	0.0007
denschnb	2	0	9	11	0	3.721375e-17	10	7	0.0007
denschnc	2	0	10	12	0	2.177679e-20	11	11	0.0012
denschnd	3	0	35	42	0	5.818610e-08	36	32	0.0029
denschne	3	0	9	11	0	1.000000e+00	10	10	0.0005
denschnf	2	0	6	8	0	6.513246e-22	7	7	0.0010
dipigri	7	4	9	60	0	6.806300e+02	15	6	0.0024
disc2	28	23	27	791	-8	0.000000e+00	41	17	0.0225
discs	36	69	13	5465	-8	1.372472e+01	20	8	0.1716
dittert	327	264	2	287	-21	-2.000000e+00	3	3	0.0223
dixchlng	10	5	10	56	0	2.471897e+03	12	11	0.0035
dixchlnv	100	50	17	542	0	3.018525e-22	18	18	0.0627
dixmaana	3000	0	9	7631	0	1.000000e+00	10	7	0.6238
dixmaanb	3000	0	14	11018	0	1.000000e+00	15	10	1.2844
dixmaanc	3000	0	13	11012	0	1.000000e+00	14	9	1.3088
dixmaand	3000	0	7	11543	-22	1.175238e+04	8	4	1.2354
dixmaane	3000	0	24	10262	0	1.000000e+00	25	14	0.8539
dixmaanf	3000	0	34	14038	0	1.000000e+00	35	17	1.7386

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
dixmaang	3000	0	10	14321	-3	1.444361e+06	11	6	1.5736
dixmaanh	3000	0	57	15947	0	1.000000e+00	58	30	1.8970
dixmaani	3000	0	20	12032	0	1.000000e+00	21	13	0.9950
dixmaanj	3000	0	10	13127	-3	6.045509e+05	11	6	1.4484
dixmaank	3000	0	62	16189	-3	1.465817e+01	63	32	2.0279
dixmaanl	3000	0	96	20460	0	1.000000e+00	97	49	2.7458
dixon3dq	10	0	1	11	0	3.204747e-30	2	2	0.0003
djtl	2	0	29	38	0	-8.951544e+03	30	14	0.0011
dnieper	61	24	3	189	0	1.874401e+04	4	4	0.0059
dqdrtic	5000	0	1	5001	0	0.000000e+00	2	2	0.4679
dqrtic	5000	0	38	14974	0	2.110693e-05	39	39	2.5647
drcavty3	10816	816	0	11645	-21	0.000000e+00	1	1	39.4230
dtoc1l	14985	9990	5	44979	0	1.253381e+02	6	6	76.5233
dtoc1na	1485	990	5	4466	0	1.270202e+01	7	6	1.0079
dtoc1nb	1485	990	6	4668	0	1.593777e+01	11	6	1.0908
dtoc1nc	1485	990	45	7245	0	2.496981e+01	90	23	2.6630
dtoc1nd	735	490	94	4641	0	1.277580e+01	187	48	1.6355
dtoc2	5994	3996	7	20072	0	5.086762e-01	9	7	16.6841
dtoc3	14997	9998	3	74975	0	2.352624e+02	4	4	123.5581
dtoc4	14997	9998	3	43196	0	2.868538e+00	4	4	68.5464
dtoc5	9998	4999	3	19998	0	1.535102e+00	4	4	14.0880
dtoc6	10000	5000	11	64751	0	1.348506e+05	12	12	51.1794
dual1	85	1	1	140	0	3.501296e-02	2	2	0.0069
dual2	96	1	1	99	0	3.373367e-02	2	2	0.0077
dual3	111	1	1	118	0	1.357558e-01	2	2	0.0127
dual4	75	1	1	65	0	7.460906e-01	2	2	0.0037
dualc1	9	215	1	16	0	6.155251e+03	2	2	0.0021
dualc2	7	229	1	19	0	3.551306e+03	2	2	0.0021
dualc5	8	278	1	13	0	4.272325e+02	2	2	0.0021
dualc8	8	503	1	29	0	1.830936e+04	2	2	0.0049

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
edensch	2000	0	7	2007	0	1.200328e+04	8	8	0.1242
eg1	3	0	7	12	0	-1.429306e+00	8	7	0.0004
eg2	1000	0	3	1003	0	-9.989473e+02	4	4	0.0651
eg3	101	200	4	207	0	6.717997e-02	5	5	0.0248
eigena	110	0	20	85	0	5.614295e-21	21	15	0.0113
eigena2	110	55	2	345	0	4.992010e-30	3	3	0.0221
eigenaco	110	55	78	1999	0	1.585725e-12	137	41	0.4643
eigenals	110	0	137	1349	0	6.544500e-18	138	74	0.4718
eigenb	110	0	79	1929	0	9.724754e-10	80	50	0.2666
eigenb2	110	55	2	402	0	1.800000e+01	3	3	0.0253
eigenbco	110	55	59	2690	-21	9.001730e+00	107	25	0.3697
eigenbls	110	0	79	1929	0	9.724759e-10	80	50	0.2901
eigenc2	462	231	347	4010	0	4.025962e-18	692	173	11.5089
eigencco	30	15	42	472	0	1.656932e-20	80	21	0.0268
eigmaxa	101	101	11	514	0	-3.000000e+00	13	11	0.0577
eigmaxb	101	101	7	1197	0	-9.674353e-04	8	8	0.0902
eigmaxc	22	22	5	110	0	-1.000000e+00	6	6	0.0038
eigmina	101	101	2	404	0	1.000000e+00	3	3	0.0633
eigminb	101	101	7	796	0	9.674353e-04	8	8	0.0693
eigminc	22	22	5	111	0	1.000000e+00	6	6	0.0040
engval1	5000	0	7	5007	0	5.548668e+03	8	8	0.8787
engval2	3	0	13	19	0	1.122731e+02	14	10	0.0007
errinros	50	0	21	118	-3	6.458821e+01	22	14	0.0032
expfit	2	0	14	21	0	2.405105e-01	15	7	0.0006
expfita	5	22	12	70	0	1.136611e-03	13	13	0.0016
expfitb	5	102	14	521	0	5.019365e-03	15	15	0.0170
expfitc	5	502	14	1686	0	2.330257e-02	15	15	0.0873
explin	120	0	12	139	0	-7.237562e+05	13	13	0.0021
explin2	120	0	11	134	0	-7.244591e+05	12	12	0.0020
expquad	120	10	12	471	0	-3.624599e+06	14	9	0.0091

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
extrasim	2	1	1	5	0	1.000000e+00	2	2	0.0005
fccu	19	8	3	83	0	1.114910e+01	4	4	0.0016
fletcbv2	100	0	1	101	0	-5.140067e-01	2	2	0.0027
fletchcr	100	0	10	110	0	6.449151e-18	11	8	0.0037
fletcher	4	5	3000	6021	-2	4.000000e+00	3001	3001	0.1380
flosp2hh	691	0	1	853	-3	8.704390e+05	2	1	0.1119
flosp2hl	691	0	3	747	0	3.887054e+01	4	4	0.0948
flosp2hm	691	0	2	820	-3	2.731203e+03	3	3	0.1054
flosp2th	691	0	7	755	0	1.000000e+01	8	8	0.1229
flosp2tl	691	0	1	811	0	1.000000e+01	2	2	0.0971
flosp2tm	691	0	2	788	-3	1.000000e+01	3	3	0.0998
fminsrf2	1024	0	43	4893	0	1.000000e+00	44	22	0.7452
fminsurf	1024	0	71	9354	0	1.000000e+00	72	35	101.7804
freuroth	5000	0	10	5012	0	6.081591e+05	11	8	0.8713
gausselm	1495	3962	1	115214	-24	-1.000000e+00	3	1	78.0569
genhs28	10	8	1	27	0	9.271736e-01	2	2	0.0012
genhumps	5	0	25	57	-3	1.657565e+04	26	16	0.0024
genrose	500	0	1010	2493	0	1.000000e+00	1011	757	0.7759
gigomez1	3	3	3	14	0	-3.000000e+00	5	3	0.0005
gilbert	1000	1	18	3779	0	4.820272e+02	20	19	0.2892
goffin	51	50	2	193	0	1.314504e-13	3	3	0.0127
gottfr	2	2	7	27	0	0.000000e+00	11	6	0.0007
gouldqp2	699	349	1	1279	0	1.879984e-04	2	2	0.1086
gouldqp3	699	349	1	1146	0	2.065155e+00	2	2	0.1392
gpp	250	498	5	2038	0	1.440092e+04	9	6	0.7304
gridneta	13284	6724	1	27828	0	3.049829e+02	2	2	24.9178
gridnetb	13284	6724	1	49596	0	1.433232e+02	2	2	63.5907
gridnetc	7564	3844	1	24237	0	1.618702e+02	2	2	16.0570
gridnetd	7565	3844	3	20439	0	5.664443e+02	4	4	19.1548
gridnete	7565	3844	3	26845	0	2.065546e+02	4	4	25.5768

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
gridnetf	7565	3844	3	24224	0	2.421089e+02	4	4	30.8966
gridnetg	61	36	4	130	0	7.331702e+01	5	5	0.0052
gridneth	61	36	4	136	0	3.962626e+01	5	5	0.0057
gridneti	61	36	4	151	0	4.024746e+01	5	5	0.0058
growth	3	0	67	104	0	1.004040e+00	68	48	0.0034
growthls	3	0	67	104	0	1.004040e+00	68	48	0.0029
gulf	3	0	25	43	0	1.044496e-17	26	18	0.0019
hadamals	100	0	11	388	-3	6.669565e+01	12	12	0.0202
hadamard	65	256	1	484	0	1.000000e+00	2	2	0.0417
hager1	10001	5001	1	16841	0	8.807970e-01	2	2	14.7071
hager2	10000	5000	1	20306	0	4.320822e-01	2	2	16.9382
hager3	10000	5000	1	20306	0	1.409612e-01	2	2	18.1063
hager4	10000	5000	1	17402	0	2.794030e+00	2	2	15.3574
haifam	85	150	1	100125	-24	-5.398098e+00	3	2	3.2544
haifas	7	9	29	281	0	-4.500000e-01	56	14	0.0052
hairy	2	0	19	30	-3	4.690335e+02	20	7	0.0007
haldmads	6	42	13	554	-21	3.311440e-01	23	6	0.0093
hanging	300	180	16	5325	-21	-2.067989e+02	33	7	0.4112
harkerp2	100	0	4	5	0	-5.000000e-01	5	5	0.0041
hart6	6	0	1	5	-3	-4.081494e-01	2	1	0.0003
hatflda	4	0	8	18	0	3.442013e-11	9	6	0.0006
hatfldb	4	1	8	24	0	5.572809e-03	10	6	0.0006
hatfldc	4	3	4	17	0	1.137789e-20	5	5	0.0010
hatfldd	3	0	13	15	0	1.404407e+01	14	14	0.0009
hatflde	3	0	12	14	0	1.528320e+01	13	13	0.0007
hatfldf	3	3	15	82	0	0.000000e+00	27	9	0.0020
hatfldg	25	25	12	248	0	0.000000e+00	17	9	0.0108
hatfldh	4	7	1	29	0	-2.450000e+01	2	2	0.0006
heart6	6	6	207	811	0	0.000000e+00	407	104	0.0340
heart6ls	6	0	28	96	-3	3.038264e+01	29	14	0.0016

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
heart8	8	8	14	298	0	0.000000e+00	22	10	0.0051
heart8ls	8	0	16	77	-3	7.749085e+01	17	8	0.0014
helix	3	0	24	39	0	5.714551e-15	25	18	0.0011
hilberta	10	0	1	42	0	7.979703e-07	2	2	0.0005
hilbertb	50	0	1	51	0	1.445383e-27	2	2	0.0017
himmelba	2	2	1	5	0	0.000000e+00	2	2	0.0002
himmelbb	2	0	100	101	0	0.000000e+00	101	55	0.0024
himmelbc	2	2	6	19	0	0.000000e+00	8	6	0.0010
himmelbd	2	2	3000	6024	-2	0.000000e+00	3010	2997	0.1530
himmelbe	3	3	2	8	0	0.000000e+00	3	3	0.0003
himmelbf	4	0	7	13	0	3.185717e+02	8	8	0.0006
himmelbg	2	0	9	11	0	2.549820e-17	10	6	0.0005
himmelbh	2	0	7	10	0	-9.999999e-01	8	5	0.0004
himmelbi	100	12	15	365	0	-1.754999e+03	16	16	0.0070
himmelbj	45	16	58	325	-4	-1.906112e+03	114	4	0.0278
himmelbk	24	14	4	29	0	5.181434e-02	5	5	0.0014
himmelp1	2	2	7	9	0	-5.173784e+01	8	8	0.0004
himmelp2	2	3	8	15	0	-6.205386e+01	9	9	0.0006
himmelp3	2	4	4	11	0	-5.901312e+01	5	5	0.0008
himmelp4	2	5	4	11	0	-5.901312e+01	5	5	0.0004
himmelp5	2	5	13	36	0	-5.901312e+01	14	11	0.0010
himmelp6	2	5	1	1	0	-5.901312e+01	2	2	0.0002
hong	4	1	7	22	0	1.347306e+00	8	8	0.0007
hs001	2	1	31	47	0	7.034748e-15	32	25	0.0013
hs002	2	1	8	17	0	4.941229e+00	9	8	0.0005
hs003	2	1	1	2	0	0.000000e+00	2	2	0.0002
hs004	2	2	2	6	0	2.666666e+00	3	3	0.0006
hs005	2	2	7	32	0	-1.913222e+00	8	6	0.0007
hs006	2	1	1	6	0	4.930380e-32	3	2	0.0003
hs007	2	1	16	36	0	-1.732050e+00	25	11	0.0010

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs008	2	2	5	9	0	-1.000000e+00	6	6	0.0005
hs009	2	1	4	6	0	-4.999999e-01	5	4	0.0004
hs010	2	1	8	11	0	-1.000000e+00	9	9	0.0007
hs011	2	1	5	8	0	-8.498464e+00	6	6	0.0005
hs012	2	1	9	28	0	-3.000000e+01	17	7	0.0008
hs013	2	1	18	36	-8	1.001353e+00	19	19	0.0011
hs014	2	2	5	10	0	1.393464e+00	6	6	0.0005
hs015	2	3	6	53	0	3.065000e+02	8	6	0.0010
hs016	2	4	6	39	0	2.314466e+01	7	7	0.0012
hs017	2	4	10	40	0	1.000000e+00	11	10	0.0010
hs018	2	4	6	23	0	5.000000e+00	9	7	0.0007
hs019	2	4	5	21	0	-6.961813e+03	6	6	0.0006
hs020	2	4	6	35	0	4.019872e+01	7	7	0.0007
hs021	2	3	1	8	0	-9.995999e+01	2	2	0.0003
hs022	2	2	4	8	0	9.999999e-01	5	5	0.0004
hs023	2	5	5	28	0	2.000000e+00	6	6	0.0006
hs024	2	3	3	3	-3	-1.336458e-02	4	1	0.0003
hs026	3	1	17	30	0	1.232329e-10	21	15	0.0011
hs027	3	1	49	125	0	3.999999e-02	95	27	0.0032
hs028	3	1	1	5	0	3.944304e-31	2	2	0.0003
hs029	3	1	5	11	-3	-1.000000e+00	8	1	0.0004
hs030	3	4	1	6	0	1.000000e+00	2	2	0.0003
hs031	3	4	5	13	0	5.999999e+00	6	6	0.0005
hs032	3	2	1	9	0	1.000000e+00	2	2	0.0003
hs033	3	3	4	5	0	-3.999999e+00	5	5	0.0003
hs034	3	5	10	35	0	-8.340328e-01	20	8	0.0009
hs035	3	1	1	5	0	1.111111e-01	2	2	0.0003
hs036	3	4	2	2	-3	-1.000000e+03	3	1	0.0002
hs037	3	2	2	2	-3	-1.000000e+03	3	1	0.0002
hs038	4	0	8	12	-3	7.877189e+00	9	9	0.0007

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs039	4	2	12	55	0	-1.000000e+00	18	10	0.0011
hs040	4	3	3	19	0	-2.500009e-01	4	4	0.0006
hs041	4	5	2	2	0	2.000000e+00	3	3	0.0003
hs042	4	2	5	13	0	1.385786e+01	6	6	0.0005
hs043	4	3	9	68	0	-4.400000e+01	17	7	0.0013
hs044	4	6	1	3	0	-1.499999e+01	2	2	0.0003
hs046	5	2	33	105	0	6.332185e-08	64	20	0.0032
hs047	5	3	21	110	0	3.131472e-09	38	16	0.0025
hs048	5	2	1	9	0	3.648481e-30	2	2	0.0003
hs049	5	2	15	23	0	6.962487e-09	16	16	0.0010
hs050	5	3	8	31	0	1.078143e-19	9	9	0.0009
hs051	5	3	1	12	0	3.574525e-31	2	2	0.0003
hs052	5	3	1	11	0	5.326647e+00	2	2	0.0003
hs053	5	3	1	12	0	4.093023e+00	2	2	0.0003
hs054	6	1	1	9	0	1.928571e-01	2	2	0.0003
hs055	6	6	2	17	0	6.333333e+00	3	3	0.0005
hs056	7	4	2	12	-3	-1.000000e+00	5	1	0.0004
hs057	2	3	3	11	0	2.845966e-02	5	4	0.0005
hs059	2	3	9	28	0	-6.749505e+00	10	9	0.0008
hs060	3	1	6	10	0	3.256820e-02	7	7	0.0005
hs061	3	2	6	25	0	-1.436461e+02	8	7	0.0007
hs062	3	1	6	18	0	-2.627251e+04	7	6	0.0007
hs063	3	2	3000	6017	-2	9.680000e+02	3002	3001	0.1179
hs064	3	1	15	59	0	6.299842e+03	20	16	0.0018
hs065	3	4	5	23	0	9.535288e-01	7	6	0.0006
hs066	3	5	7	30	0	5.181632e-01	14	5	0.0009
hs067	10	21	34	149	0	-1.162026e+03	65	22	0.0068
hs070	4	1	13	27	0	1.795701e-01	15	10	0.0014
hs071	4	2	5	10	0	1.701401e+01	6	6	0.0005
hs072	4	6	9	73	0	7.276788e+02	11	10	0.0014

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs073	4	3	2	17	0	2.989437e+01	3	3	0.0006
hs074	4	4	38	119	0	5.126498e+03	52	25	0.0031
hs075	4	4	38	127	0	5.174412e+03	52	25	0.0033
hs076	4	3	1	8	0	-4.681818e+00	2	2	0.0003
hs077	5	2	10	24	0	2.415051e-01	14	10	0.0009
hs078	5	3	4	26	0	-2.919700e+00	5	5	0.0007
hs079	5	3	4	16	0	7.877682e-02	5	5	0.0006
hs080	5	3	7	18	0	5.394984e-02	8	8	0.0007
hs081	5	3	15	66	0	4.388512e-01	24	11	0.0018
hs083	5	3	4	20	0	-3.066553e+04	5	5	0.0006
hs084	5	3	0	4	-23	-2.351243e+06	1	1	0.0004
hs085	5	48	10	26	0	-1.905155e+00	17	9	0.0019
hs086	5	10	3	34	0	-3.234867e+01	4	4	0.0009
hs087	11	6	11	88	0	8.827597e+03	12	12	0.0018
hs088	2	1	16	42	0	1.362656e+00	19	15	0.0022
hs089	3	1	22	95	0	1.362656e+00	30	17	0.0037
hs090	4	1	40	212	0	1.362656e+00	70	19	0.0070
hs091	5	1	21	81	0	1.362656e+00	28	16	0.0047
hs092	6	1	33	165	0	1.362656e+00	57	19	0.0088
hs093	6	2	10	58	0	1.350759e+02	20	6	0.0014
hs095	6	4	2	3	0	1.561952e-02	3	3	0.0003
hs096	6	4	2	3	0	1.561952e-02	3	3	0.0003
hs097	6	4	5	15	0	3.135809e+00	7	6	0.0006
hs098	6	4	5	15	0	3.135809e+00	7	6	0.0005
hs099	23	18	16	247	0	-8.310798e+08	17	17	0.0066
hs100	7	4	9	60	0	6.806300e+02	15	6	0.0011
hs100lnp	7	2	11	45	0	6.806300e+02	17	9	0.0012
hs100mod	7	4	8	42	0	6.787547e+02	15	5	0.0012
hs101	7	6	20	219	0	1.809764e+03	26	16	0.0034
hs102	7	6	16	199	0	9.118805e+02	22	11	0.0034

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs103	7	6	14	195	0	5.436679e+02	21	11	0.0031
hs104	8	6	14	93	0	3.951163e+00	22	8	0.0021
hs105	8	9	10	53	0	1.151396e+03	11	11	0.0087
hs106	8	14	1716	4020	0	7.049248e+03	3431	859	0.1667
hs107	9	14	21	379	0	5.055011e+03	34	15	0.0080
hs108	9	14	8	247	-21	-6.718773e-01	14	4	0.0035
hs109	9	10	23	183	0	5.326851e+03	31	18	0.0039
hs110	10	0	4	34	0	-4.577846e+01	5	4	0.0006
hs111	10	3	23	276	0	-4.737064e+01	37	17	0.0046
hs111lnp	10	3	23	276	0	-4.737064e+01	37	17	0.0045
hs112	10	3	11	73	0	-4.776109e+01	12	12	0.0017
hs113	10	8	5	33	0	2.430620e+01	6	6	0.0009
hs114	10	11	27	110	0	-1.768806e+03	49	19	0.0046
hs116	13	28	175	496	0	9.758750e+01	347	89	0.0381
hs117	15	5	5	24	0	3.234867e+01	6	6	0.0008
hs118	15	17	2	26	0	6.648204e+02	3	3	0.0006
hs119	16	8	7	97	0	2.448996e+02	8	8	0.0026
hs21mod	7	7	1	20	0	-9.595999e+01	2	2	0.0004
hs268	5	5	1	14	0	7.219114e-12	2	2	0.0004
hs35mod	3	2	1	6	0	2.500000e-01	2	2	0.0006
hs3mod	2	1	1	4	0	0.000000e+00	2	2	0.0002
hs44new	4	6	1	3	0	-1.500000e+01	2	2	0.0003
hs99exp	31	21	54	535	0	-1.008062e+09	92	38	0.0197
hubfit	2	1	1	4	0	1.689349e-02	2	2	0.0004
hues-mod	10000	2	13	65465	0	3.482448e+07	14	14	55.9720
huestis	10000	2	11	57403	-8	8.358408e+10	12	12	47.7188
humps	2	0	30	44	-3	2.138289e+04	31	18	0.0009
hvycrash	202	150	12	103239	-24	-2.254036e-01	20	7	2.0446
hypcir	2	2	6	20	0	0.000000e+00	8	6	0.0005
integreq	100	100	2	202	0	0.000000e+00	3	3	0.0438

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
jensmp	2	0	9	11	0	1.243621e+02	10	10	0.0006
kissing	127	903	25	14723	-22	1.413553e+00	50	11	2.2163
kiwcresc	3	2	12	38	0	-6.776352e-08	20	8	0.0011
kowosb	4	0	17	33	0	3.075056e-04	18	9	0.0010
ksip	20	1001	1	1148	0	5.757979e-01	2	2	0.0997
lakes	90	78	11	1104	-23	6.069344e+11	12	12	0.0411
launch	25	29	48	838	-3	9.512672e+00	79	17	0.0191
lch	600	1	0	505	-21	2.586969e+05	1	1	0.0284
liarwhd	10000	0	12	10012	0	8.198347e-22	13	13	8.4733
linspanh	97	33	1	72	0	-7.699999e+01	2	2	0.0031
liswet1	10002	10000	1	26769	0	3.612061e+01	2	2	23.1894
liswet10	10002	10000	1	24357	0	4.948391e+01	2	2	20.7017
liswet11	10002	10000	1	24497	0	4.951524e+01	2	2	20.8351
liswet12	10002	10000	1	27270	0	-3.314380e+03	2	2	24.1918
liswet2	10002	10000	2	21232	0	2.499996e+01	3	3	16.4498
liswet3	10002	10000	1	19836	0	2.499980e+01	2	2	14.5672
liswet4	10002	10000	1	19824	0	2.499980e+01	2	2	14.6010
liswet5	10002	10000	1	19838	0	2.499981e+01	2	2	14.5093
liswet6	10002	10000	1	19897	0	2.499986e+01	2	2	14.6920
liswet7	10002	10000	1	25994	0	4.987922e+02	2	2	22.0970
liswet8	10002	10000	1	25807	0	7.144874e+02	2	2	22.1193
liswet9	10002	10000	1	28041	0	1.963305e+03	2	2	24.9196
lminsurf	15625	496	0	2774	-23	1.000000e+00	1	1	1.5957
loadbal	31	31	7	51	0	4.528510e-01	8	8	0.0018
loghairy	2	0	6	7	-3	6.552519e+00	7	1	0.0003
logros	2	0	11	12	-3	1.275930e+00	12	2	0.0004
lootsma	3	3	4	5	0	2.000000e+00	5	5	0.0004
lotschd	12	7	2	49	0	2.398415e+03	3	3	0.0010
lsnnodoc	5	4	6	51	0	1.231124e+02	7	7	0.0010
lsqfit	2	1	1	4	0	3.378698e-02	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
madsen	3	6	13	70	0	6.164324e-01	23	8	0.0015
madsschj	81	158	783	7191	0	-7.972837e+02	1546	391	4.2405
makela1	3	2	15	75	0	-1.414213e+00	29	6	0.0014
makela2	3	3	9	47	0	7.199999e+00	15	7	0.0009
makela3	21	20	54	355	0	-1.387778e-17	103	28	0.0076
makela4	21	40	2	35	0	0.000000e+00	3	3	0.0009
mancino	100	0	5	127	0	1.677174e-21	6	6	0.0393
manne	1094	730	1	270	0	-9.745725e-01	2	2	0.0525
maratos	2	1	8	24	0	-1.000000e+00	15	5	0.0007
maratosb	2	0	211	315	0	-1.000000e+00	212	156	0.0063
matrix2	6	2	10	16	0	9.536743e-07	11	11	0.0007
maxlika	8	8	10	53	0	1.151396e+03	11	11	0.0103
mconcon	15	16	8	78	0	-6.230795e+03	13	9	0.0020
mdhole	2	0	2	3	0	0.000000e+00	3	3	0.0003
methanb8	31	0	72	256	0	4.784765e-20	73	39	0.0178
methanl8	31	0	361	1889	-3	9.540468e-03	362	181	0.0949
mexhat	2	0	9	11	0	-4.009999e-02	10	7	0.0005
meyer3	3	0	416	554	0	8.794585e+01	417	344	0.0230
mifflin1	3	2	10	41	0	-1.000000e+00	21	6	0.0015
mifflin2	3	2	10	54	0	-1.000000e+00	18	6	0.0021
minc44	311	262	3000	11603	-2	2.575683e-03	5996	1496	27.1996
minmaxbd	5	20	62	448	0	1.157064e+02	121	31	0.0099
minmaxrb	3	4	1	7	0	0.000000e+00	3	2	0.0004
minperm	1113	1033	4	229	-21	2.499999e-01	7	4	0.0599
minsurf	64	32	1	125	0	1.000000e+00	2	2	0.0047
mistake	9	13	18	247	0	-1.000000e+00	35	7	0.0048
model	1831	339	1	185	0	5.742163e+03	2	2	0.0278
mosarqp1	2500	700	1	1762	0	-9.528754e+02	2	2	0.1887
mosarqp2	900	600	1	1064	0	-1.597482e+03	2	2	0.0666
msqrtals	1024	0	106	24552	0	3.615234e-11	107	51	177.7508

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
msqrtbls	1024	0	86	23658	0	3.595141e-11	87	41	145.8149
mwright	5	3	9	38	0	2.497880e+01	11	8	0.0011
nasty	2	0	1	3	0	1.534091e-72	2	2	0.0002
ncvxbqp1	10000	0	1	9584	0	-1.985543e+10	2	2	6.3145
ncvxbqp2	10000	0	1	8673	0	-1.329056e+10	2	2	5.4847
ncvxbqp3	10000	0	1	5169	0	-6.470378e+09	2	2	3.0478
ncvxqp1	1000	500	1	7085	0	-7.163867e+07	2	2	1.2096
ncvxqp2	1000	500	1	8131	0	-5.780691e+07	2	2	1.4146
ncvxqp3	1000	500	1	7552	0	-3.123876e+07	2	2	1.1894
ncvxqp4	1000	250	1	2756	0	-9.398212e+07	2	2	0.3329
ncvxqp5	1000	250	1	2688	0	-6.633766e+07	2	2	0.3201
ncvxqp6	1000	250	1	2271	0	-3.515589e+07	2	2	0.2604
ncvxqp7	1000	750	1	14621	0	-4.352453e+07	2	2	3.7517
ncvxqp8	1000	750	1	15438	0	-3.049666e+07	2	2	4.1419
ncvxqp9	1000	750	1	16038	0	-2.157081e+07	2	2	3.7392
ngone	100	1273	71	1197	0	-6.376367e-01	143	33	0.2989
noncvxu2	1000	0	215	324645	-24	2.431469e+04	216	108	48.5667
noncvxun	1000	0	81	7766	0	2.656813e+03	82	43	0.2145
nondia	9999	0	5	10006	0	2.306591e-25	6	6	4.2736
nondquar	10000	0	19	10021	0	4.139781e-10	20	20	7.4667
nonmsqrt	9	0	235	398	-3	7.518036e-01	236	138	0.0116
nonscomp	10000	0	8	10010	0	3.059438e-14	9	9	2.7079
nuffield	861	8001	2	130355	-21	-2.697031e+00	5	1	115.9462
nuffield2	2382	7923	0	100000	-24	-0.000000e+00	1	1	115.6763
nuffield_continuum	2	1	3	7	0	-2.549414e+00	4	4	0.0009
obstclal	96	0	1	38	0	1.397897e+00	2	2	0.0018
obstclbl	96	0	1	73	0	2.875038e+00	2	2	0.0021
obstclbu	96	0	1	57	0	2.875038e+00	2	2	0.0019
odfits	10	6	9	52	0	-2.380026e+03	10	10	0.0026
oet1	3	1002	1	313	0	5.382431e-01	2	2	0.0333

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
oet2	3	1002	7	3408	0	8.715964e-02	10	6	0.3574
oet3	4	1002	1	264	0	4.505052e-03	2	2	0.0296
oet7	7	1002	68	12685	0	2.069736e-03	132	31	1.4050
optcdeg2	1199	800	3	18380	0	2.295734e+02	4	4	3.5370
optcdeg3	1199	800	3	13734	0	4.614566e+01	4	4	2.2344
optcntrl	32	21	2	158	0	5.500000e+02	3	3	0.0060
optctrl3	122	81	0	711	-8	1.560050e+06	1	1	0.0458
optctrl6	122	81	0	711	-8	1.560050e+06	1	1	0.0471
optmass	66	55	8	457	-3	-1.468523e-01	15	3	0.0215
optprloc	30	30	6	136	0	-1.641977e+01	13	5	0.0056
orthrdm2	4003	2000	5	13076	0	1.555328e+02	7	6	50.7529
orthrds2	203	100	24	107143	-24	1.397629e+03	36	14	4.4093
orthrega	517	256	62	12771	0	1.414055e+03	119	36	2.7787
orthregb	27	6	1	44	0	1.449963e-28	2	2	0.0007
orthrege	36	20	796	1934	0	6.071141e-01	1590	400	0.2846
osbornea	5	0	3000	5294	-2	4.695101e-02	3001	1500	0.1295
osborneb	11	0	18	42	0	4.013773e-02	19	13	0.0022
oslbqp	8	8	1	26	0	6.250000e+00	2	2	0.0005
palmer1	4	0	32	48	0	1.175460e+04	37	19	0.0013
palmer1a	6	0	49	84	0	8.988362e-02	50	34	0.0025
palmer1b	4	0	30	42	0	3.447354e+00	31	21	0.0015
palmer1c	8	0	6	38	0	9.759799e-02	7	7	0.0008
palmer1d	7	0	5	26	0	6.526825e-01	6	6	0.0006
palmer1e	8	0	11	45	-3	3.570721e+00	12	9	0.0010
palmer2	4	0	35	63	0	3.651089e+03	36	19	0.0013
palmer2a	6	0	75	121	0	1.716073e-02	76	62	0.0033
palmer2b	4	0	21	33	0	6.233946e-01	22	16	0.0011
palmer2c	8	0	4	30	0	1.442139e-02	5	5	0.0006
palmer2e	8	0	6	36	0	1.163139e-01	7	6	0.0011
palmer3	4	0	31	57	0	2.265958e+03	32	16	0.0013

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
palmer3a	6	0	82	143	0	2.043142e-02	83	66	0.0041
palmer3b	4	0	15	30	-3	2.346901e+02	16	10	0.0008
palmer3c	8	0	4	36	0	1.953763e-02	5	5	0.0006
palmer3e	8	0	4	22	-3	1.507276e-01	5	5	0.0006
palmer4	4	0	37	72	0	2.285383e+03	38	19	0.0028
palmer4a	6	0	15	56	-3	2.051506e+00	16	10	0.0010
palmer4b	4	0	24	46	0	6.835138e+00	25	18	0.0010
palmer4c	8	0	5	37	0	5.031069e-02	6	6	0.0006
palmer4e	8	0	92	292	-3	3.712711e-01	93	48	0.0054
palmer5a	8	0	119	226	-3	2.146426e-01	124	84	0.0049
palmer5b	9	0	53	129	-3	1.146864e-02	58	37	0.0029
palmer5c	6	0	3	15	0	2.128086e+00	4	4	0.0004
palmer5d	4	0	4	12	0	8.733939e+01	5	5	0.0007
palmer5e	8	0	4	18	-3	2.961811e+00	5	5	0.0006
palmer6a	6	0	138	248	0	5.594883e-02	151	101	0.0065
palmer6c	8	0	5	41	0	1.638742e-02	6	6	0.0007
palmer6e	8	0	16	61	-3	1.292492e-01	17	10	0.0012
palmer7a	6	0	25	59	-3	2.874189e+01	26	26	0.0020
palmer7c	8	0	7	35	0	6.019856e-01	8	8	0.0006
palmer7e	8	0	8	44	-3	5.170740e+02	9	8	0.0007
palmer8a	6	0	25	61	-3	7.719709e+00	26	26	0.0026
palmer8c	8	0	6	40	0	1.597680e-01	7	7	0.0007
palmer8e	8	0	427	1065	-3	6.340525e-01	428	214	0.0173
penalty1	1000	0	47	4389	0	9.686340e-03	48	44	60.8410
penalty2	100	0	19	119	0	9.709608e+04	20	20	0.0463
pentagon	6	15	7	60	0	1.462141e-04	8	5	0.0017
pentdi	1000	0	1	3	0	-7.500000e-01	2	2	0.0015
pfit1	3	0	14	23	-3	1.355458e+02	15	8	0.0008
pfit1ls	3	0	14	23	-3	1.355458e+02	15	8	0.0006
pfit2	3	0	14	21	-3	6.310268e+01	15	7	0.0006

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
pfit2ls	3	0	14	21	-3	6.310268e+01	15	7	0.0006
pfit3	3	0	88	125	0	5.566516e-21	89	66	0.0033
pfit3ls	3	0	88	125	0	5.566516e-21	89	66	0.0033
pfit4	3	0	382	603	0	2.371073e-20	383	269	0.0125
pfit4ls	3	0	382	603	0	2.371073e-20	383	269	0.0150
polak1	3	2	7	13	0	2.718281e+00	8	8	0.0007
polak2	11	2	3000	3004	-2	-2.971073e+13	3001	3001	0.1154
polak3	12	10	9	148	0	5.933002e+00	16	7	0.0026
polak4	3	3	4	7	0	-2.953681e-16	5	5	0.0009
polak5	3	2	43	137	0	5.000000e+01	77	29	0.0033
polak6	5	4	318	1736	0	-4.399999e+01	616	157	0.0279
porous1	4900	4900	0	721	-21	0.000000e+00	1	1	0.5241
portfl1	12	1	1	11	0	2.048627e-02	2	2	0.0004
portfl2	12	1	1	12	0	2.968923e-02	2	2	0.0004
portfl3	12	1	1	11	0	3.274970e-02	2	2	0.0004
portfl4	12	1	1	11	0	2.630695e-02	2	2	0.0004
portfl6	12	1	1	11	0	2.579179e-02	2	2	0.0004
powell20	1000	1000	5	4981	0	5.214578e+07	6	6	0.5891
powellbs	2	2	24	77	0	0.000000e+00	43	17	0.0017
powellsq	2	2	62	220	0	0.000000e+00	114	32	0.0043
power	1000	0	1	1001	0	0.000000e+00	2	2	0.0455
probpenl	500	0	19	519	0	-4.850917e-07	20	20	0.4013
prodpl0	60	29	10	321	0	6.091923e+01	12	9	0.0097
prodpl1	60	29	8	278	0	5.303701e+01	10	7	0.0069
pspdoc	4	1	12	23	0	2.414213e+00	13	8	0.0007
pt	2	501	1	391	0	1.783942e-01	2	2	0.0213
qpcboei1	384	348	8	4424	0	1.443386e+07	9	9	0.3655
qpcboei2	143	140	7	1490	0	8.293665e+06	8	8	0.0648
qpcstair	467	356	2	2054	-21	2.141900e+05	3	3	0.2400
qpnboei1	384	348	7	4678	0	8.449924e+06	8	8	0.3877

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
qpnboei2	143	140	8	1415	0	1.271825e+06	9	9	0.0621
qpnstair	467	356	9	3246	0	5.146033e+06	10	10	0.3427
qr3d	155	0	388	1270	0	4.967249e-15	389	195	1.5728
qr3dbd	127	0	100	346	0	5.433610e-14	101	51	0.2662
qr3dls	155	0	388	1270	0	4.967383e-15	389	195	1.6389
qrtquad	120	10	16	490	0	-3.648088e+06	18	12	0.0108
quartc	10000	0	41	29977	0	2.620441e-05	42	42	8.4409
qudlin	12	0	1	13	0	-7.200000e+03	2	2	0.0002
reading1	10001	5000	3	124140	-24	-7.019024e-02	6	2	115.9778
reading2	15003	10002	2	26494	0	-1.258021e-02	3	3	52.0719
reading3	202	102	2	255	0	-6.702236e-32	3	3	0.0106
recipe	3	3	2	8	0	0.000000e+00	3	3	0.0003
rk23	17	11	13	147	0	8.333333e-02	20	9	0.0043
robot	14	9	6	62	0	1.339073e+01	8	7	0.0014
rosenbr	2	0	28	40	0	4.260063e-22	29	22	0.0010
rosenmmx	5	4	132	496	0	-4.400000e+01	264	64	0.0123
s332	2	102	11	785	0	2.992435e+01	19	9	0.0183
s365mod	7	9	889	1908	-8	2.500000e-01	898	887	0.0514
sawpath	593	786	0	4918	-21	1.483193e+03	1	1	0.9174
scon1dls	1000	0	313	17267	0	1.606123e-11	314	239	1.0497
scosine	10000	0	20	6983	-22	1.804312e+03	21	2	3.5914
scurly10	10000	0	30	83297	0	-9.945407e+05	31	23	84.8311
scurly20	10000	0	32	34338	0	-9.966678e+05	33	25	57.4581
scurly30	10000	0	32	36788	0	-9.974118e+05	33	26	92.6072
semicon1	1000	1000	41	5723	-8	0.000000e+00	62	22	1.4579
semicon2	1000	1000	21	8177	-8	0.000000e+00	32	12	1.9594
sim2bqp	2	0	1	3	0	0.000000e+00	2	2	0.0002
simbqp	2	1	1	4	0	-2.220446e-16	2	2	0.0002
simpllpa	2	2	1	3	0	1.000000e+00	2	2	0.0002
simpllpb	2	3	1	3	0	1.099999e+00	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
sineali	20	0	3	5	0	-1.827032e+03	4	4	0.0003
sineval	2	0	56	85	0	1.409206e-18	57	42	0.0019
sinquad	10000	0	4	10004	-3	1.474312e-02	5	5	11.8170
sipow1	2	10000	1	972	0	-1.000000e+00	2	2	0.7433
sipow1m	2	10000	1	971	0	-1.000000e+00	2	2	0.7194
sipow2	2	5001	1	185	0	-1.000000e+00	2	2	0.0736
sipow2m	2	5001	1	182	0	-1.000000e+00	2	2	0.0752
sipow3	4	10000	1	224	0	5.356507e-01	2	2	0.1802
sipow4	4	10000	1	4858	0	2.728283e-01	2	2	3.8269
sisser	2	0	16	18	0	9.640368e-09	17	13	0.0006
smbank	117	64	20	452	0	-7.129292e+06	21	21	0.0344
smmpsf	720	263	22	2383	0	1.046985e+06	23	23	0.2723
snake	2	2	2	15	0	-2.373656e-13	3	3	0.0010
sosqp1	20000	10001	1	38587	0	-5.249689e-13	2	2	54.4829
sosqp2	20000	10001	1	37297	0	-4.998699e+03	2	2	70.7995
spanhyd	97	33	31	1644	-3	2.398075e+02	36	15	0.0566
spiral	3	2	173	788	0	-2.798963e-11	326	105	0.0195
sreadin3	10001	5001	1	5001	0	0.000000e+00	2	2	6.0966
srosenbr	10000	0	28	60028	0	2.130031e-18	29	22	11.4345
sseblin	194	72	10	993	0	1.617060e+07	11	11	0.0488
ssebnln	194	96	47	1762	0	1.617060e+07	67	29	0.0988
ssnlbeam	33	20	4	262	0	3.377724e+02	5	5	0.0064
stancmin	3	2	5	16	0	4.250000e+00	6	6	0.0005
steenbra	432	108	8	1040	0	1.695767e+04	9	9	0.0762
steenbrb	468	108	484	23594	0	9.075855e+03	485	250	1.4593
steenbrc	540	126	162	2648	0	1.837538e+04	189	90	0.2141
steenbrd	468	108	404	21485	0	9.144724e+03	406	211	1.2803
steenbre	540	126	1228	86628	0	2.745916e+04	1229	624	5.6801
steenbrf	468	108	69	1767	0	2.826795e+02	78	40	0.1014
steenbrg	540	126	1479	111404	0	2.742092e+04	1483	750	7.3592

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
supersim	2	2	1	4	0	6.666666e-01	2	2	0.0002
svanberg	5000	5000	3	58941	-3	9.165666e+03	5	2	41.5320
swopf	83	92	4	294	0	6.786014e-02	5	5	0.0244
synthes1	7	7	4	24	0	5.171320e+00	9	5	0.0009
tame	2	1	1	5	0	1.232595e-32	2	2	0.0003
tfi2	3	10001	1	883	0	6.490420e-01	2	2	0.6826
tointqor	50	0	1	53	0	1.175472e+03	2	2	0.0008
trainf	20008	10002	3	79612	0	3.103384e+00	4	4	174.1029
tridia	10000	0	1	10001	0	3.430769e-24	2	2	2.7102
trimloss	142	75	7	255	0	9.060000e+00	8	8	0.0142
try-b	2	1	9	18	0	0.000000e+00	11	9	0.0014
twirism1	343	313	8	22827	-21	-1.780800e-01	15	6	3.1104
twobars	2	2	11	31	0	1.508652e+00	20	9	0.0020
ubh1	17997	12000	0	100000	-24	0.000000e+00	1	1	126.2072
vanderm1	100	199	10	181396	-24	0.000000e+00	21	4	12.9324
vanderm2	100	199	16	26345	-8	0.000000e+00	28	9	3.6439
vanderm3	100	199	3	9549	-21	0.000000e+00	6	2	1.1143
vanderm4	9	17	48	437	-8	0.000000e+00	65	33	0.0139
vardim	100	0	25	125	0	0.000000e+00	26	26	0.0558
watson	31	0	10	301	-3	7.358741e-04	11	11	0.0046
weeds	3	0	23	30	0	9.205435e+03	24	23	0.0010
womflet	3	3	8	24	0	0.000000e+00	12	6	0.0008
woods	10000	0	8	10008	-3	1.969297e+04	9	9	2.1469
yao	2002	2003	1	9974	0	1.977046e+02	2	2	2.5958
yfit	3	0	46	60	0	6.669720e-13	47	39	0.0027
yfitu	3	0	46	60	0	6.669720e-13	47	39	0.0023
zangwil2	2	0	1	3	0	-1.819999e+01	2	2	0.0003
zangwil3	3	3	4	22	0	0.000000e+00	5	5	0.0006
zecevic2	2	4	1	8	0	-4.125000e+00	2	2	0.0003
zecevic3	2	4	45	165	0	9.730945e+01	89	23	0.0029

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
zecevic4	2	4	5	20	0	7.557507e+00	6	6	0.0006
zigzag	64	50	8	301	0	3.161734e+00	11	8	0.0120
zy2	3	2	4	5	0	2.000000e+00	5	5	0.0004

B.2. Warm-start Using Perturbed Optimal Solution

Table B.2. Table of warm-start results on the CUTE test set for RestartSQP

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
3pk	30	0	1	1	0	1.720118+00	2	2	0.0006
aircrfta	5	5	1	1	0	0.000000+00	2	2	0.0003
aircrftb	8	3	1	1	0	8.000173-13	2	2	0.0003
airport	84	42	4	13	0	4.795270+04	5	5	0.0052
aljazzaf	3	1	1	1	0	7.500499+01	2	2	0.0002
allinit	4	3	2	2	0	1.670596+01	3	3	0.0004
allinitc	4	4	2	3	0	3.049413+01	3	3	0.0002
allinitu	4	0	1	1	0	5.744384+00	2	2	0.0002
alsotame	2	3	1	2	0	8.208499-02	2	2	0.0003
arglina	100	0	1	1	0	9.999999+01	2	2	0.0135
arglinb	10	0	1	22	-3	4.642266+00	2	1	0.0004
arglinc	8	0	2	107393	-24	6.145743+00	3	1	0.3783
argtrig	100	100	2	2	0	0.000000+00	3	3	0.0134
arwhead	5000	0	45	5044	-4	-1.519877-09	46	7	1.8024
aug2d	20192	9996	1	1	0	1.687411+06	2	2	0.1094
aug2dc	20200	10194	1	30	0	1.818392+06	2	2	0.2110
aug2dcqp	20200	10194	1	15	0	6.498179+06	2	2	0.1519
aug2dqp	20192	10194	1	1	0	6.237011+06	2	2	0.1157
aug3d	3873	1000	1	6268	0	5.540677+02	2	2	2.4905
aug3dc	3873	1000	1	1	0	7.712624+02	2	2	0.0205

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
aug3dcqp	3873	1000	1	56	0	9.933621+02	2	2	0.0406
aug3dqp	3873	1000	1	8	0	6.752376+02	2	2	0.0388
avgasa	8	10	1	2	0	-4.412171+00	2	2	0.0003
avgasb	8	10	1	2	0	-4.483219+00	2	2	0.0003
avion2	49	15	1	22	-3	9.468023+07	2	1	0.0007
bard	3	0	1	1	0	8.214877-03	2	2	0.0002
batch	46	73	2	26	0	2.591803+05	3	3	0.0011
bdexp	5000	0	0	0	0	1.324840-06	1	1	0.0014
bdqrtic	1000	0	2	2	0	3.983817+03	3	3	0.0085
bdvalue	5000	5000	1	1	0	0.000000+00	2	2	0.0191
beale	2	0	0	0	0	4.342569-18	1	1	0.0001
bigbank	2230	1112	1	12	0	-4.205696+06	2	2	0.0350
biggs3	6	3	1	1	0	3.181446-13	2	2	0.0002
biggs5	6	1	1	1	0	1.449749-10	2	2	0.0002
biggs6	6	0	1	1	0	1.312266-10	2	2	0.0003
biggsb1	1000	999	1	1	0	1.499999-02	2	2	0.0028
biggsc4	4	7	1	10	0	-2.450000+01	2	2	0.0004
blockqp1	2005	1001	1	1	-3	-9.964767+02	2	1	0.0687
blockqp2	2005	1001	1	49	0	-9.961011+02	2	2	0.1483
blockqp3	2005	1001	1	1	-3	-4.974846+02	2	1	0.0712
blockqp4	2005	1001	1	57	0	-4.980982+02	2	2	0.1933
blockqp5	2005	1001	1	1	-3	-4.974846+02	2	1	0.0706
bloweya	2002	1002	1	1	0	-4.553071-02	2	2	0.0966
bloweyb	2002	1002	1	1	0	-3.045265-02	2	2	0.0940
bloweyc	2002	1002	1	1	0	-3.036381-02	2	2	0.0983
booth	2	2	1	1	0	0.000000+00	2	2	0.0003
box2	3	1	1	1	0	3.862337-33	2	2	0.0002
box3	3	0	1	1	0	6.185938-30	2	2	0.0002
bqp1var	1	1	0	0	0	-7.479521-09	1	1	0.0001
bqpgabim	50	4	1	3	0	-3.790343-05	2	2	0.0006

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
bqpgasim	50	0	1	3	0	-5.519814-05	2	2	0.0005
brainpc7	6905	6900	0	277	-23	3.927065-04	2	1	162.8901
bratu1d	1001	0	1	1	0	-8.518927+00	2	2	0.0032
bratu2d	4900	4900	1	1	0	0.000000+00	2	2	0.0849
bratu3d	3375	3375	1	1	0	0.000000+00	2	2	0.2432
brkmcc	2	0	0	0	0	1.690426-01	1	1	0.0001
brownal	10	0	1	1	0	1.116698-13	2	2	0.0003
brownbs	2	0	0	0	0	0.000000+00	1	1	0.0001
brownden	4	0	2	2	0	8.582220+04	3	3	0.0003
broydn3d	10000	10000	1	1	0	0.000000+00	2	2	0.0470
broydn7d	1000	0	2	2	0	3.450050+02	3	3	0.0055
broydnbd	5000	5000	2	2	0	0.000000+00	3	3	0.0803
brybnd	5000	0	2	2	0	8.021454-18	3	3	0.0597
bt1	2	1	1	1	0	-1.000000+00	2	2	0.0002
bt10	2	2	1	1	0	-1.000000+00	2	2	0.0002
bt11	5	3	1	1	0	8.248917-01	2	2	0.0003
bt12	5	3	1	1	0	6.188118+00	2	2	0.0003
bt13	5	2	0	0	0	-7.494096-09	1	1	0.0001
bt2	3	1	1	1	0	3.256824-02	2	2	0.0003
bt3	5	3	1	1	0	4.093023+00	2	2	0.0003
bt4	3	2	1	1	0	-3.704768+00	2	2	0.0003
bt5	3	2	1	1	0	9.617151+02	2	2	0.0003
bt6	5	2	1	1	0	2.770450-01	2	2	0.0003
bt7	5	3	1	1	0	3.064997+02	2	2	0.0003
bt8	5	2	0	0	0	1.000000+00	1	1	0.0001
bt9	4	2	1	1	0	-1.000000+00	2	2	0.0003
byrdsphr	3	2	1	1	0	-4.683300+00	2	2	0.0003
camel6	2	2	0	0	0	-2.154638-01	1	1	0.0001
cantilvr	5	1	1	1	0	1.339956+00	2	2	0.0002
catena	32	11	1	1	0	-2.307775+04	2	2	0.0004

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
catenary	496	166	1	1	0	-3.484032+05	2	2	0.0015
cb2	3	3	1	1	0	1.952224+00	2	2	0.0003
cb3	3	3	1	1	0	2.000000+00	2	2	0.0003
cbratu2d	882	882	1	1	0	0.000000+00	2	2	0.0124
cbratu3d	1024	1024	1	1	0	0.000000+00	2	2	0.0397
chaconn1	3	3	1	1	0	1.952224+00	2	2	0.0002
chaconn2	3	3	1	1	0	2.000000+00	2	2	0.0003
chainwoo	1000	0	2	2	0	6.362471+01	3	3	0.0031
chandheq	100	100	1	1	0	0.000000+00	2	2	0.0147
chebyqad	50	0	13	22	0	5.386315-03	14	10	0.2125
chemrctb	1000	1000	1	1	0	0.000000+00	2	2	0.0048
chenhark	1000	0	1	11	0	-2.000000+00	2	2	0.0022
chnrosnb	50	0	2	2	0	1.582934-16	3	3	0.0004
cliff	2	0	4	4	0	1.997866-01	5	5	0.0003
clnlbeam	1499	1000	3	27331	0	3.448762+02	4	4	6.1623
clplatea	4970	0	2	2	0	-1.259209-02	3	3	0.0432
clplateb	4970	0	2	2	0	-6.988222+00	3	3	0.0461
clplatec	4970	0	1	1	0	-5.020724-03	2	2	0.0253
cluster	2	2	1	1	0	0.000000+00	2	2	0.0002
concon	15	11	1	2	0	-6.230795+03	2	2	0.0003
congigmz	3	5	1	1	0	2.799999+01	2	2	0.0003
coolhans	9	9	1	1	0	0.000000+00	2	2	0.0003
core1	65	115	1	809	0	9.105624+01	3	2	0.0481
core2	157	134	0	100000	-24	7.290065+01	1	1	3.2597
corkscrw	8997	7000	0	100000	-24	9.068793+01	1	1	145.8280
coshfun	61	20	0	19	-8	9.712808-08	1	1	0.0017
cosine	10000	0	2	2	0	-9.999000+03	3	3	0.0347
cragglvy	5000	0	2	2	0	1.688215+03	3	3	0.0207
cresc4	6	8	2	7	0	8.718975-01	3	3	0.0005
csfi1	5	4	1	1	0	-4.907520+01	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
csfi2	5	4	0	0	0	5.501760+01	1	1	0.0001
cube	2	0	0	0	0	1.753567-24	1	1	0.0001
curly10	10000	0	2	2	0	-1.003162+06	3	3	0.1245
curly20	10000	0	2	2	0	-1.003162+06	3	3	0.2657
curly30	10000	0	2	2	0	-1.003162+06	3	3	0.4341
cvxbqp1	10000	0	1	1	0	2.250225+06	2	2	0.0164
cvxqp1	1000	500	1	11	0	1.087511+06	2	2	0.0277
cvxqp2	10000	2500	1	79	0	8.184245+07	2	2	0.7256
dallass	46	31	0	103	-21	-3.239350+04	1	1	0.0058
deconvc	51	1	41	191	0	2.569477-03	69	19	0.0228
deconvu	51	0	4	4	-3	5.730984-03	5	5	0.0025
demymalo	3	3	1	1	0	-2.999999+00	2	2	0.0003
denschna	2	0	0	0	0	1.102837-23	1	1	0.0001
denschnb	2	0	0	0	0	9.860761-32	1	1	0.0001
denschnc	2	0	0	0	0	2.177679-20	1	1	0.0001
denschnd	3	0	9	9	0	1.222451-08	10	10	0.0006
denschne	3	0	1	1	0	2.227301-13	2	2	0.0002
denschnf	2	0	0	0	0	6.513246-22	1	1	0.0001
dipigri	7	4	1	2	0	6.806300+02	2	2	0.0003
disc2	28	23	15	220	0	1.562500+00	31	5	0.0108
discs	36	69	0	1337	-8	1.444961+01	1	1	0.0450
dittert	327	264	1	1	0	-1.997596+00	2	2	0.0089
dixchlng	10	5	2	2	0	2.471897+03	3	3	0.0005
dixchlnv	100	50	2	2	0	2.419489-22	3	3	0.0049
dixmaana	3000	0	1	1	0	1.000000+00	2	2	0.0062
dixmaanb	3000	0	1	1	0	1.000000+00	2	2	0.0110
dixmaanc	3000	0	1	1	0	1.000000+00	2	2	0.0108
dixmaand	3000	0	1	1	0	1.000000+00	2	2	0.0120
dixmaane	3000	0	1	1	0	1.000000+00	2	2	0.0066
dixmaanf	3000	0	1	1	0	1.000000+00	2	2	0.0120

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
dixmaang	3000	0	1	1	0	1.000000+00	2	2	0.0115
dixmaanh	3000	0	1	1	0	1.000000+00	2	2	0.0117
dixmaani	3000	0	1	1	0	1.000000+00	2	2	0.0062
dixmaanj	3000	0	1	1	0	1.000000+00	2	2	0.0116
dixmaank	3000	0	1	1	0	1.000000+00	2	2	0.0115
dixmaanl	3000	0	1	1	0	1.000000+00	2	2	0.0117
dixon3dq	10	0	1	1	0	0.000000+00	2	2	0.0003
djtl	2	0	1	1	0	-8.951544+03	2	2	0.0003
dnieper	61	24	1	1	0	1.874401+04	2	2	0.0006
dqdrtic	5000	0	1	1	0	2.509036-34	2	2	0.0058
dqrtic	5000	0	43	44	0	1.380912-08	44	15	0.0665
dtoc1l	14985	9990	1	1	0	1.253381+02	2	2	0.1180
dtoc1na	1485	990	1	1	0	1.270202+01	2	2	0.0357
dtoc1nb	1485	990	1	1	0	1.593777+01	2	2	0.0350
dtoc1nc	1485	990	1	1	0	2.496981+01	2	2	0.0356
dtoc1nd	735	490	1	1	0	1.275156+01	2	2	0.0137
dtoc2	5994	3996	1	1	0	5.086765-01	2	2	0.0379
dtoc3	14997	9998	1	1	0	2.352624+02	2	2	0.0659
dtoc4	14997	9998	1	1	0	2.868538+00	2	2	0.0666
dtoc5	9998	4999	1	1	0	1.535111+00	2	2	0.0333
dtoc6	10000	5000	1	1	0	1.348506+05	2	2	0.0360
dual1	85	1	1	6	0	3.501296-02	2	2	0.0063
dual2	96	1	1	1	0	3.373367-02	2	2	0.0072
dual3	111	1	1	1	0	1.357558-01	2	2	0.0073
dual4	75	1	1	3	0	7.460906-01	2	2	0.0037
dualc1	9	215	1	1	0	6.155251+03	2	2	0.0009
dualc2	7	229	1	1	0	3.551306+03	2	2	0.0009
dualc5	8	278	1	1	0	4.272325+02	2	2	0.0010
dualc8	8	503	1	9	0	1.830936+04	2	2	0.0022
edensch	2000	0	2	2	0	1.200328+04	3	3	0.0079

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
eg1	3	0	1	3	0	-1.429306+00	2	2	0.0003
eg2	1000	0	2	2	0	-9.989473+02	3	3	0.0179
eg3	101	200	1	196	0	1.540743-33	3	2	0.0312
eigena	110	0	2	27	0	1.283811-16	3	3	0.0090
eigena2	110	55	2	2	0	1.544785-23	3	3	0.0039
eigenaco	110	55	2	2	0	8.999177-17	3	3	0.0126
eigenals	110	0	2	2	0	1.425016-15	3	3	0.0130
eigenb	110	0	2	2	0	3.963127-12	3	3	0.0095
eigenb2	110	55	0	295	-21	1.799603+01	2	1	0.0259
eigenbco	110	55	2	2	0	5.735917-18	3	3	0.0128
eigenbls	110	0	2	2	0	3.962938-12	3	3	0.0129
eigenc2	462	231	2	2	0	8.014228-20	3	3	0.0430
eigencco	30	15	1	1	0	1.023484-11	2	2	0.0008
eigmaxb	101	101	1	1	0	-9.674354-04	2	2	0.0016
eigmaxc	22	22	1	1	0	-1.000000+00	2	2	0.0005
eigminb	101	101	1	1	0	9.674354-04	2	2	0.0017
engval1	5000	0	2	2	0	5.548668+03	3	3	0.0183
engval2	3	0	2	2	0	1.537610-18	3	3	0.0003
errinros	50	0	3	3	0	4.040449+01	4	4	0.0006
expfit	2	0	0	0	0	2.405105-01	1	1	0.0001
expfita	5	22	1	1	0	1.136611-03	2	2	0.0004
expfitb	5	102	1	1	0	5.019365-03	2	2	0.0006
explin	120	0	0	0	0	-7.237562+05	1	1	0.0001
explin2	120	0	0	0	0	-7.244591+05	1	1	0.0001
expquad	120	10	1	25	0	-3.624599+06	2	2	0.0014
extrasim	2	1	1	1	0	1.000000+00	2	2	0.0002
extrosnb	10	0	60	85	0	3.151063-12	61	38	0.0034
fccu	19	8	1	1	0	1.114910+01	2	2	0.0004
fletcbv2	100	0	1	1	0	-5.140067-01	2	2	0.0005
fletchcr	100	0	2	2	0	6.717228-17	3	3	0.0007

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
fletcher	4	5	1	2	0	1.165685+01	2	2	0.0003
flosp2hl	691	0	8	367	-3	6.462626+01	9	2	0.1547
flosp2hm	691	0	1	100134	-24	1.397378+06	2	1	31.0094
flosp2th	691	0	2	10	-3	1.114218+09	3	3	0.0220
flosp2tl	691	0	1	2	-3	1.564988+02	2	1	0.0141
flosp2tm	691	0	1	5	-3	1.397349+06	2	1	0.0150
fminsrf2	1024	0	1	1	0	1.000000+00	2	2	0.0067
fminsurf	1024	0	1	1	0	1.000000+00	2	2	2.1780
freuroth	5000	0	2	2	0	6.081591+05	3	3	0.0246
genhs28	10	8	1	1	0	9.271736-01	2	2	0.0003
genhumps	5	0	1	1	0	3.703859-28	2	2	0.0003
genrose	500	0	2	2	0	1.000000+00	3	3	0.0021
gigomez1	3	3	1	1	0	-3.000000+00	2	2	0.0003
gilbert	1000	1	2	2	0	4.820272+02	3	3	0.0067
goffin	51	50	2	187	-8	2.954266-05	3	3	0.0277
gottfr	2	2	1	1	0	0.000000+00	2	2	0.0002
gouldqp2	699	349	0	168	-21	1.995334-04	1	1	0.0212
gouldqp3	699	349	1	11	0	2.065155+00	2	2	0.0026
gpp	250	498	2	13	0	1.440092+04	3	3	0.0835
gridneta	13284	6724	1	40	0	3.049829+02	2	2	0.1049
gridnetb	13284	6724	1	1	0	1.433232+02	2	2	0.0890
gridnetc	7564	3844	1	160	0	1.618702+02	2	2	0.2176
gridnetd	7565	3844	2	8579	0	5.664443+02	3	3	10.5912
gridnete	7565	3844	1	14518	0	2.065546+02	2	2	23.9609
gridnetf	7565	3844	0	1355	-21	2.421200+02	1	1	15.9704
gridnetg	61	36	1	5	0	7.331702+01	2	2	0.0010
gridneth	61	36	1	279	0	3.962626+01	2	2	0.0052
gridneti	61	36	1	1	0	4.024746+01	2	2	0.0007
growth	3	0	3	3	0	1.004040+00	4	4	0.0004
growthls	3	0	3	3	0	1.004040+00	4	4	0.0004

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
gulf	3	0	2	2	0	2.139958-17	3	3	0.0004
hadamals	100	0	2	5	0	2.531641+01	3	3	0.0033
hager1	10001	5001	1	1	0	8.807970-01	2	2	0.0291
hager2	10000	5000	1	1	0	4.320822-01	2	2	0.0362
hager3	10000	5000	1	1	0	1.409612-01	2	2	0.0451
hager4	10000	5000	1	2376	0	2.794030+00	2	2	2.2396
haifam	85	150	0	109	-21	-4.500036+01	1	1	0.0073
haifas	7	9	4	76	0	-4.499999-01	9	2	0.0014
hairy	2	0	0	0	0	2.000000+01	1	1	0.0001
haldmads	6	42	3	7	0	3.303040-02	4	4	0.0008
hanging	300	180	4	547	0	-6.201760+02	5	5	0.0664
harkerp2	100	0	1	40	0	-5.000000-01	2	2	0.0037
hart6	6	0	2	2	0	-3.322886+00	3	3	0.0003
hatflda	4	0	1	1	0	6.584295-14	2	2	0.0003
hatfldb	4	1	1	2	0	5.572809-03	2	2	0.0003
hatfldc	4	3	1	1	0	1.289489-12	2	2	0.0003
hatfldd	3	0	1	1	0	6.692488-08	2	2	0.0003
hatflde	3	0	1	1	0	4.438232-07	2	2	0.0003
hatfldf	3	3	1	1	0	0.000000+00	2	2	0.0003
hatfldg	25	25	1	1	0	0.000000+00	2	2	0.0005
hatfldh	4	7	1	10	0	-2.450000+01	2	2	0.0004
heart6	6	6	1	1	0	0.000000+00	2	2	0.0003
heart6ls	6	0	3	3	0	2.579327-28	4	4	0.0004
heart8	8	8	1	1	0	0.000000+00	2	2	0.0004
heart8ls	8	0	2	2	0	1.205648-15	3	3	0.0004
helix	3	0	1	1	0	1.271711-29	2	2	0.0003
hilberta	10	0	1	1	0	9.727142-28	2	2	0.0003
hilbertb	50	0	1	1	0	1.315952-36	2	2	0.0014
himmelba	2	2	1	1	0	0.000000+00	2	2	0.0002
himmelbb	2	0	0	0	0	2.126863-21	1	1	0.0001

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
himmelbc	2	2	1	1	0	0.000000+00	2	2	0.0003
himmelbe	3	3	1	1	0	0.000000+00	2	2	0.0002
himmelbf	4	0	1	1	0	3.185717+02	2	2	0.0003
himmelbg	2	0	0	0	0	3.632999-22	1	1	0.0001
himmelbh	2	0	0	0	0	-1.000000+00	1	1	0.0001
himmelbi	100	12	0	0	0	-1.754999+03	1	1	0.0001
himmelp1	2	2	0	0	0	-6.205386+01	1	1	0.0001
himmelp2	2	3	1	1	0	-8.198031+00	2	2	0.0003
himmelp3	2	4	0	0	0	-5.901312+01	1	1	0.0001
himmelp4	2	5	0	0	0	-5.901312+01	1	1	0.0001
himmelp5	2	5	0	0	0	-5.901312+01	1	1	0.0001
himmelp6	2	5	0	0	0	-5.901312+01	1	1	0.0001
hong	4	1	1	1	0	1.347306+00	2	2	0.0002
hs001	2	1	0	0	0	5.894625-16	1	1	0.0001
hs002	2	1	1	1	-3	4.941229+00	2	1	0.0002
hs003	2	1	1	1	0	0.000000+00	2	2	0.0002
hs004	2	2	1	1	0	2.666666+00	2	2	0.0002
hs005	2	2	0	0	0	-1.913222+00	1	1	0.0001
hs006	2	1	1	1	0	0.000000+00	2	2	0.0002
hs007	2	1	1	1	0	-1.732050+00	2	2	0.0003
hs008	2	2	1	1	0	-1.000000+00	2	2	0.0003
hs009	2	1	1	1	0	-4.999999-01	2	2	0.0003
hs010	2	1	1	1	0	-1.000000+00	2	2	0.0003
hs011	2	1	1	1	0	-8.498464+00	2	2	0.0003
hs012	2	1	1	1	0	-3.000000+01	2	2	0.0003
hs013	2	1	0	0	0	9.945785-01	1	1	0.0001
hs014	2	2	1	1	0	1.393464+00	2	2	0.0003
hs015	2	3	1	1	0	3.064999+02	2	2	0.0002
hs016	2	4	0	0	0	2.314466+01	1	1	0.0001
hs017	2	4	2	3	0	1.000000+00	3	3	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs018	2	4	1	1	0	5.000000+00	2	2	0.0003
hs019	2	4	1	1	0	-6.961813+03	2	2	0.0003
hs020	2	4	0	0	0	4.019872+01	1	1	0.0001
hs021	2	3	1	4	0	-9.995999+01	2	2	0.0003
hs022	2	2	1	1	0	1.000000+00	2	2	0.0003
hs023	2	5	0	0	0	1.999999+00	1	1	0.0001
hs024	2	3	1	1	0	-1.000000+00	2	2	0.0003
hs025	3	3	2	2	0	2.084767-17	3	3	0.0005
hs026	3	1	1	1	0	8.398750-15	2	2	0.0003
hs027	3	1	1	1	0	4.000000-02	2	2	0.0003
hs028	3	1	1	1	0	0.000000+00	2	2	0.0003
hs029	3	1	1	1	-3	-2.262107+01	2	1	0.0003
hs030	3	4	1	1	0	1.000000+00	2	2	0.0002
hs031	3	4	1	1	0	6.000000+00	2	2	0.0003
hs032	3	2	1	1	0	1.000000+00	2	2	0.0003
hs033	3	3	1	1	0	-4.585786+00	2	2	0.0003
hs034	3	5	1	2	0	-8.340324-01	2	2	0.0003
hs035	3	1	1	2	0	1.111111-01	2	2	0.0003
hs036	3	4	1	2	0	-3.299999+03	2	2	0.0003
hs037	3	2	1	1	-3	-3.455838+03	2	1	0.0002
hs038	4	0	2	2	0	2.523899-16	3	3	0.0003
hs039	4	2	1	1	0	-1.000000+00	2	2	0.0003
hs040	4	3	1	1	0	-2.500001-01	2	2	0.0003
hs041	4	5	1	1	-3	1.926050+00	2	1	0.0003
hs042	4	2	1	1	0	1.385786+01	2	2	0.0003
hs043	4	3	2	4	0	-4.400000+01	3	3	0.0003
hs044	4	6	1	2	0	-1.300000+01	2	2	0.0003
hs045	5	0	1	2	0	1.000000+00	2	2	0.0002
hs046	5	2	1	1	0	4.554769-15	2	2	0.0003
hs047	5	3	1	1	0	2.700098-11	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs048	5	2	1	1	0	0.000000+00	2	2	0.0004
hs049	5	2	1	1	0	2.093819-12	2	2	0.0003
hs050	5	3	1	1	0	2.659116-19	2	2	0.0003
hs051	5	3	1	1	0	2.465190-32	2	2	0.0003
hs052	5	3	1	1	0	5.326647+00	2	2	0.0003
hs053	5	3	1	1	0	4.093023+00	2	2	0.0004
hs054	6	1	1	1	0	1.928571-01	2	2	0.0003
hs055	6	6	2	8	0	6.333333+00	3	3	0.0005
hs056	7	4	1	1	0	-3.455999+00	2	2	0.0003
hs057	2	3	0	0	0	3.064761-02	1	1	0.0001
hs059	2	3	1	1	0	-7.802789+00	2	2	0.0003
hs060	3	1	1	1	0	3.256824-02	2	2	0.0003
hs061	3	2	1	1	0	-1.436461+02	2	2	0.0003
hs062	3	1	3	3	0	-2.627251+04	4	4	0.0004
hs063	3	2	1	1	0	9.617151+02	2	2	0.0003
hs064	3	1	0	0	0	6.299838+03	1	1	0.0001
hs065	3	4	2	3	0	9.535288-01	3	3	0.0003
hs066	3	5	1	1	0	5.181632-01	2	2	0.0003
hs067	10	21	1	2	0	-1.162026+03	2	2	0.0004
hs070	4	1	1	1	0	9.401973-03	2	2	0.0003
hs071	4	2	1	1	0	1.701401+01	2	2	0.0003
hs072	4	6	0	0	0	7.276783+02	1	1	0.0001
hs073	4	3	1	1	0	2.989437+01	2	2	0.0003
hs074	4	4	1	1	0	5.126498+03	2	2	0.0003
hs075	4	4	1	1	0	5.174412+03	2	2	0.0003
hs076	4	3	1	2	0	-4.681818+00	2	2	0.0003
hs077	5	2	1	1	0	2.415051-01	2	2	0.0003
hs078	5	3	1	1	0	-2.919700+00	2	2	0.0003
hs079	5	3	1	1	0	7.877686-02	2	2	0.0003
hs080	5	3	1	1	0	5.394983-02	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs081	5	3	1	1	0	5.394983-02	2	2	0.0003
hs083	5	3	2	4	0	-3.066553+04	3	3	0.0004
hs084	5	3	1	1	0	-5.280335+06	2	2	0.0002
hs085	5	48	4	5	0	-1.905155+00	5	5	0.0009
hs086	5	10	1	5	0	-3.234867+01	2	2	0.0004
hs087	11	6	2	13	0	8.827597+03	3	3	0.0006
hs088	2	1	0	0	0	1.362646+00	1	1	0.0002
hs089	3	1	2	2	0	1.362656+00	3	3	0.0007
hs090	4	1	2	2	0	1.362656+00	3	3	0.0008
hs091	5	1	2	2	0	1.362656+00	3	3	0.0010
hs092	6	1	2	2	0	1.362656+00	3	3	0.0013
hs093	6	2	2	3	0	1.350759+02	3	3	0.0004
hs095	6	4	1	1	0	1.561952-02	2	2	0.0002
hs096	6	4	1	1	0	1.561952-02	2	2	0.0002
hs097	6	4	2	4	0	3.135809+00	3	3	0.0004
hs098	6	4	2	4	0	3.135809+00	3	3	0.0003
hs099	23	18	2	2	0	-8.310798+08	3	3	0.0007
hs100	7	4	1	2	0	6.806300+02	2	2	0.0003
hs100lnp	7	2	1	1	0	6.806300+02	2	2	0.0003
hs100mod	7	4	1	2	0	6.787547+02	2	2	0.0003
hs101	7	6	11	83	0	1.809764+03	21	5	0.0020
hs102	7	6	9	74	0	9.118805+02	18	5	0.0020
hs103	7	6	3	12	0	5.436679+02	4	4	0.0006
hs104	8	6	8	45	0	3.951163+00	17	4	0.0016
hs105	8	9	1	1	0	1.136360+03	2	2	0.0020
hs106	8	14	717	1575	0	7.049248+03	1432	356	0.0901
hs107	9	14	1	1	0	5.055011+03	2	2	0.0003
hs108	9	14	0	2	-21	-6.747588-01	1	1	0.0005
hs109	9	10	2	2	0	5.326851+03	3	3	0.0004
hs110	10	0	1	1	0	-4.577846+01	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs111	10	3	1	1	0	-4.776109+01	2	2	0.0004
hs111lnp	10	3	1	1	0	-4.776109+01	2	2	0.0004
hs112	10	3	1	1	0	-4.776109+01	2	2	0.0003
hs113	10	8	2	4	0	2.430620+01	3	3	0.0005
hs114	10	11	1	1	0	-1.768806+03	2	2	0.0003
hs116	13	28	8	69	0	9.758750+01	16	5	0.0024
hs117	15	5	2	20	0	3.234867+01	3	3	0.0006
hs118	15	17	1	5	0	6.648204+02	2	2	0.0004
hs119	16	8	1	1	0	2.448996+02	2	2	0.0004
hs21mod	7	7	1	4	0	-9.595999+01	2	2	0.0003
hs268	5	5	1	1	0	3.092281-11	2	2	0.0003
hs35mod	3	2	1	1	0	2.500000-01	2	2	0.0003
hs3mod	2	1	1	1	0	0.000000+00	2	2	0.0002
hs44new	4	6	1	2	0	-1.500000+01	2	2	0.0003
hs99exp	31	21	1	1	0	-1.008062+09	2	2	0.0003
hubfit	2	1	1	2	0	1.689349-02	2	2	0.0003
hues-mod	10000	2	1	28	0	3.482448+07	2	2	0.3728
huestis	10000	2	0	0	0	3.482448+11	1	1	0.0016
humps	2	0	0	0	0	6.178161-23	1	1	0.0001
hvycrash	202	150	1	121	-21	-1.442099-01	3	2	0.0101
hypcir	2	2	1	1	0	0.000000+00	2	2	0.0003
integreq	100	100	1	1	0	0.000000+00	2	2	0.0085
jensmp	2	0	0	0	0	1.243621+02	1	1	0.0001
kissing	127	903	0	461	-21	1.000001+00	1	1	0.1140
kiwcresc	3	2	1	1	0	-1.832301-17	2	2	0.0003
kowosb	4	0	1	1	0	3.075056-04	2	2	0.0003
lakes	90	78	1	1	0	3.505247+05	2	2	0.0010
lch	600	1	1	1979	-3	-4.278012+00	3	1	1.1479
liarwhd	10000	0	2	2	0	4.085322-20	3	3	1.3782
linspanh	97	33	1	44	0	-7.700000+01	2	2	0.0017

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
liswet1	10002	10000	2	31604	0	3.612064+01	3	3	63.2918
liswet10	10002	10000	1	7721	0	4.948391+01	2	2	9.7906
liswet11	10002	10000	1	6763	0	4.951524+01	2	2	8.8242
liswet12	10002	10000	1	10199	0	-3.314380+03	2	2	12.7284
liswet2	10002	10000	3	25317	0	2.499990+01	4	4	39.6691
liswet3	10002	10000	1	8640	0	2.499980+01	2	2	9.0417
liswet4	10002	10000	1	8288	0	2.499980+01	2	2	8.8171
liswet5	10002	10000	1	4844	0	2.499981+01	2	2	6.0570
liswet6	10002	10000	1	4817	0	2.499986+01	2	2	6.0532
liswet7	10002	10000	1	6749	0	4.987922+02	2	2	8.9898
liswet8	10002	10000	1	8543	0	7.144874+02	2	2	11.4500
liswet9	10002	10000	1	10348	0	1.963305+03	2	2	14.6796
lminsurf	15625	496	1	1	0	9.000000+00	2	2	0.1718
loadbal	31	31	1	13	0	4.528510-01	2	2	0.0006
loghairy	2	0	0	0	0	1.823215-01	1	1	0.0001
logros	2	0	0	0	0	0.000000+00	1	1	0.0001
lootsma	3	3	1	1	0	1.414213+00	2	2	0.0003
lotschd	12	7	1	1	0	2.398415+03	2	2	0.0003
lsqfit	2	1	1	2	0	3.378698-02	2	2	0.0003
madsen	3	6	1	1	0	6.164324-01	2	2	0.0003
madsschj	81	158	2	257	0	-7.972837+02	3	3	0.0350
makela1	3	2	1	1	0	-1.414213+00	2	2	0.0003
makela2	3	3	1	1	0	7.199999+00	2	2	0.0003
makela3	21	20	1	12	-3	4.011808-08	3	1	0.0005
makela4	21	40	1	20	0	3.970466-23	2	2	0.0009
mancino	100	0	2	2	0	1.937943-21	3	3	0.0119
manne	1094	730	1	137	-3	-9.736259-01	3	1	0.0251
maratos	2	1	1	1	0	-1.000000+00	2	2	0.0003
maratosb	2	0	0	0	0	-1.000000+00	1	1	0.0001
matrix2	6	2	1	3	0	1.147943-41	2	2	0.0004

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
maxlika	8	8	1	1	0	1.136360+03	2	2	0.0020
mccormck	50000	0	1	1	0	-4.566161+04	2	2	0.1100
mconcon	15	16	1	2	0	-6.230795+03	2	2	0.0003
mdhole	2	0	0	0	0	1.835221-13	1	1	0.0001
methanb8	31	0	5	5	0	5.062205-15	6	6	0.0017
methanl8	31	0	5	5	0	5.062385-15	6	6	0.0017
mexhat	2	0	0	0	0	-4.010000-02	1	1	0.0001
meyer3	3	0	3000	3000	-2	8.794585+01	3001	3001	0.1313
mifflin1	3	2	1	1	0	-1.000000+00	2	2	0.0003
mifflin2	3	2	1	1	0	-1.000000+00	2	2	0.0003
minc44	311	262	1	1	0	2.572988-03	2	2	0.0085
minmaxbd	5	20	9	127	0	1.157064+02	19	3	0.0027
minperm	1113	1033	4	19042	-21	3.597274-04	9	2	10.1097
minsurf	64	32	1	1	0	1.000000+00	2	2	0.0008
mistake	9	13	1	25	0	-1.000000+00	3	2	0.0009
morebv	5002	2	1	1	0	1.704725-12	2	2	0.0139
mosarqp1	2500	700	1	287	0	-9.528754+02	2	2	0.0468
mosarqp2	900	600	1	106	0	-1.597482+03	2	2	0.0104
msqrta	1024	1024	1	1	0	0.000000+00	2	2	0.2327
msqrtals	1024	0	3	3	0	6.656775-15	4	4	5.4463
msqrtb	1024	1024	1	1	0	0.000000+00	2	2	0.2289
msqrtbls	1024	0	3	3	0	1.171856-12	4	4	5.4250
mwright	5	3	1	1	0	2.497881+01	2	2	0.0003
nasty	2	0	0	0	0	0.000000+00	1	1	0.0001
ncvxbqp1	10000	0	0	0	0	-1.985543+10	1	1	0.0017
ncvxbqp2	10000	0	1	1	0	-1.331510+10	2	2	0.0169
ncvxbqp3	10000	0	1	1	0	-6.437652+09	2	2	0.0188
ncvxqp6	1000	250	1	1	0	-3.408012+07	2	2	0.0050
ngone	100	1273	0	2	-21	-6.332900-01	1	1	0.0067
noncvxu2	1000	0	1	1	0	2.317572+03	2	2	0.0450

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
noncvxun	1000	0	1	1	0	2.316808+03	2	2	0.0015
nondia	9999	0	3	3	0	8.945414-17	4	4	2.8358
nondquar	10000	0	0	0	0	1.634974-09	1	1	0.0025
nonscomp	10000	0	2	110	0	2.253142-18	3	3	0.0962
nuffield_continuum	2	1	0	0	0	-2.549414+00	1	1	0.0001
obstclal	96	0	1	4	0	1.397897+00	2	2	0.0007
obstclbl	96	0	1	6	0	2.875038+00	2	2	0.0007
obstclbu	96	0	1	6	0	2.875038+00	2	2	0.0007
odfits	10	6	1	1	0	-2.380026+03	2	2	0.0004
optcdeg2	1199	800	1	832	0	2.295734+02	2	2	0.2144
optcdeg3	1199	800	1	1023	0	4.614566+01	2	2	0.2632
optctrl3	122	81	1	1	0	2.048016+03	2	2	0.0011
optctrl6	122	81	1	1	0	2.048016+03	2	2	0.0011
optmass	66	55	0	197	-8	-1.895424-01	1	1	0.0177
optprloc	30	30	2	19	0	-1.641977+01	3	3	0.0015
orthrdm2	4003	2000	2	2	0	1.555328+02	3	3	12.5221
orthrds2	203	100	1	226	-8	1.544304+03	3	1	0.0402
orthrega	517	256	2	2	0	1.508635+03	3	3	0.0513
orthregb	27	6	6	59	0	2.531975-16	13	4	0.0048
orthrege	36	20	64	161	0	1.517387+00	127	30	0.0478
osbornea	5	0	2	2	0	5.464894-05	3	3	0.0009
osborneb	11	0	1	1	0	4.013773-02	2	2	0.0012
oslbqp	8	8	1	1	0	6.250000+00	2	2	0.0006
palmer1	4	0	6	6	0	1.175460+04	7	7	0.0015
palmer1a	6	0	1	1	0	8.988362-02	2	2	0.0006
palmer1b	4	0	1	1	0	3.447354+00	2	2	0.0006
palmer1c	8	0	1	1	0	9.759799-02	2	2	0.0008
palmer1d	7	0	1	1	0	6.526825-01	2	2	0.0006
palmer1e	8	0	2	2	0	8.352682-04	3	3	0.0009
palmer2	4	0	5	10	0	3.651089+03	6	5	0.0013

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
palmer2a	6	0	1	1	0	1.716073-02	2	2	0.0007
palmer2b	4	0	2	2	0	6.233946-01	3	3	0.0009
palmer2c	8	0	1	1	0	1.442139-02	2	2	0.0007
palmer2e	8	0	1	1	0	2.153524-04	2	2	0.0007
palmer3	4	0	3	3	0	2.265958+03	4	4	0.0011
palmer3a	6	0	1	1	0	2.043142-02	2	2	0.0007
palmer3b	4	0	2	2	0	4.227647+00	3	3	0.0008
palmer3c	8	0	1	1	0	1.953763-02	2	2	0.0007
palmer3e	8	0	1	1	0	5.074084-05	2	2	0.0007
palmer4a	6	0	1	1	0	4.060613-02	2	2	0.0006
palmer4b	4	0	2	2	0	6.835138+00	3	3	0.0008
palmer4c	8	0	1	1	0	5.031069-02	2	2	0.0006
palmer4e	8	0	1	1	0	1.480042-04	2	2	0.0007
palmer5b	9	0	3	3	0	9.752492-03	4	4	0.0016
palmer5c	6	0	1	1	0	2.128086+00	2	2	0.0008
palmer5d	4	0	1	1	0	8.733939+01	2	2	0.0007
palmer6a	6	0	1	1	0	5.594883-02	2	2	0.0009
palmer6c	8	0	1	1	0	1.638742-02	2	2	0.0011
palmer6e	8	0	1	1	0	2.239550-04	2	2	0.0007
palmer7c	8	0	1	1	0	6.019856-01	2	2	0.0009
palmer8a	6	0	1	1	0	7.400969-02	2	2	0.0006
palmer8c	8	0	1	1	0	1.597680-01	2	2	0.0006
palmer8e	8	0	1	1	0	6.339307-03	2	2	0.0006
penalty1	1000	0	17	1403	0	9.686210-03	18	16	47.9589
penalty2	100	0	2	2	0	9.709608+04	3	3	0.0089
pentagon	6	15	1	19	-3	1.366693-04	3	1	0.0008
pentdi	1000	0	1	21	0	-7.500000-01	2	2	0.0039
pfit1	3	0	4	4	0	1.424232-16	5	5	0.0005
pfit1ls	3	0	4	4	0	1.424232-16	5	5	0.0006
pfit2	3	0	5	5	0	5.862905-17	6	6	0.0008

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
pfit2ls	3	0	5	5	0	5.862905-17	6	6	0.0007
pfit3	3	0	5	5	0	3.997588-16	6	6	0.0006
pfit3ls	3	0	5	5	0	3.997588-16	6	6	0.0007
pfit4	3	0	5	5	0	1.595551-16	6	6	0.0006
pfit4ls	3	0	5	5	0	1.595551-16	6	6	0.0007
polak1	3	2	1	1	0	2.718281+00	2	2	0.0004
polak3	12	10	2	5	0	5.933003+00	3	3	0.0007
polak4	3	3	2	4	0	2.060150-17	3	3	0.0005
polak5	3	2	1	1	0	5.000000+01	2	2	0.0004
polak6	5	4	3	5	0	-4.400000+01	4	4	0.0007
porous1	4900	4900	3	3	0	0.000000+00	4	4	0.3644
porous2	4900	4900	2	2	0	0.000000+00	3	3	0.2476
portfl1	12	1	1	4	0	2.048627-02	2	2	0.0005
portfl2	12	1	1	6	0	2.968923-02	2	2	0.0006
portfl3	12	1	1	1	0	3.274970-02	2	2	0.0004
portfl4	12	1	1	2	0	2.630695-02	2	2	0.0005
portfl6	12	1	1	4	0	2.579179-02	2	2	0.0005
powell20	1000	1000	1	204	0	5.214578+07	2	2	0.0462
powellbs	2	2	2	2	0	0.000000+00	3	3	0.0004
power	1000	0	1	1	0	7.717710-32	2	2	0.0015
probpenl	500	0	1	26	-3	9.724295-05	3	1	0.6697
prodpl0	60	29	2	38	0	6.091923+01	3	3	0.0022
prodpl1	60	29	2	33	0	5.303701+01	3	3	0.0020
pspdoc	4	1	1	1	0	2.414213+00	2	2	0.0004
pt	2	501	1	385	0	1.783942-01	2	2	0.0300
qpcboei1	384	348	1	153	0	1.443386+07	2	2	0.0205
qpcboei2	143	140	1	91	0	8.293665+06	2	2	0.0071
qpcstair	467	356	1	60	0	6.204391+06	2	2	0.0150
qpnboei1	384	348	1	153	0	8.459253+06	2	2	0.0217
qpnboei2	143	140	1	127	0	1.271825+06	2	2	0.0089

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
qpnstair	467	356	1	61	0	5.146033+06	2	2	0.0149
qr3d	155	0	3	3	0	7.153960-17	4	4	0.0279
qr3dbd	127	0	1	1	0	5.817246-11	2	2	0.0071
qr3dls	155	0	3	3	0	7.153961-17	4	4	0.0313
qrtquad	120	10	1	2	0	-3.648088+06	2	2	0.0008
quartc	10000	0	43	44	0	3.328625-08	44	16	0.1944
qudlin	12	0	0	0	0	-7.200000+03	1	1	0.0001
reading1	10001	5000	0	537	-23	-1.604670-01	1	1	1.8573
reading2	15003	10002	0	27	-21	-1.256853-02	1	1	0.2183
reading3	202	102	2	280	0	-1.022739-16	3	3	0.0370
recipe	3	3	1	1	0	0.000000+00	2	2	0.0004
res	18	14	1	1	0	0.000000+00	2	2	0.0007
rk23	17	11	1	122	-3	8.409016-02	3	1	0.0048
robot	14	9	1	1	0	1.339072+01	2	2	0.0007
rosenbr	2	0	0	0	0	3.743975-21	1	1	0.0002
rosenmmx	5	4	3	5	0	-4.400000+01	4	4	0.0007
s332	2	102	1	1	0	2.992435+01	2	2	0.0007
s365mod	7	9	2	3	0	5.213990+01	3	3	0.0007
s368	100	0	1	1	-3	-5.525068-08	2	1	0.0213
scon1dls	1000	0	2	2	0	6.249170-13	3	3	0.0062
scosine	10000	0	3000	3246	-2	-9.297359+03	3001	2988	58.0278
scurly10	10000	0	16	16	0	-9.755310+05	17	17	1.1599
scurly20	10000	0	16	16	0	-9.736919+05	17	17	2.5504
scurly30	10000	0	23	71	0	-9.753515+05	24	20	5.7870
semicon1	1000	1000	2	2	0	0.000000+00	3	3	0.0104
semicon2	1000	1000	1	1	0	0.000000+00	2	2	0.0061
sim2bqp	2	0	0	0	0	0.000000+00	1	1	0.0001
simbqp	2	1	1	1	0	0.000000+00	2	2	0.0003
simpllpa	2	2	1	1	0	1.000000+00	2	2	0.0003
simpllpb	2	3	1	1	0	1.100000+00	2	2	0.0003

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
sineval	2	0	0	0	0	5.787362-43	1	1	0.0001
sinquad	10000	0	12	41	0	9.809346-11	13	11	13.3140
sinrosnb	1000	1000	2	2	0	-9.990100+04	3	3	0.0069
sipow1	2	10000	1	2382	0	-1.000000+00	2	2	3.2359
sipow1m	2	10000	1	1	0	-1.000000+00	2	2	0.0214
sipow2	2	5001	1	1642	0	-1.000000+00	2	2	1.1911
sipow2m	2	5001	1	839	0	-1.000000+00	2	2	0.4840
sipow3	4	10000	0	0	-21	5.356507-01	1	1	0.0159
sisser	2	0	0	0	0	8.121606-13	1	1	0.0001
smbank	117	64	1	1	0	-7.129292+06	2	2	0.0018
smmpsf	720	263	1	1034	-8	1.046985+06	2	1	0.1806
snake	2	2	1	1	0	-6.082374-17	2	2	0.0003
sosqp1	20000	10001	1	1	0	3.406944-10	2	2	0.0626
sosqp2	20000	10001	1	2836	0	-4.998699+03	2	2	15.4964
spanhyd	97	33	0	103	-21	2.397392+02	1	1	0.0045
spiral	3	2	2	3	0	1.410253-13	3	3	0.0004
sreadin3	10001	5001	0	100000	-24	-6.446787-05	1	1	166.4634
srosenbr	10000	0	3	3	0	1.061785-20	4	4	0.0307
sseblin	194	72	1	262	0	1.617060+07	2	2	0.0166
ssebnln	194	96	3	195	0	1.617060+07	4	4	0.0136
ssnlbeam	33	20	1	1	0	3.377724+02	2	2	0.0006
stancmin	3	2	1	6	0	4.249999+00	2	2	0.0004
steenbrf	468	108	2	35	-3	4.614767+02	3	3	0.0046
supersim	2	2	1	1	0	6.666666-01	2	2	0.0004
svanberg	5000	5000	0	1120	-23	8.361405+03	1	1	1.1308
swopf	83	92	1	1	0	6.785976-02	2	2	0.0009
tame	2	1	1	1	0	0.000000+00	2	2	0.0004
tointqor	50	0	1	1	0	1.175472+03	2	2	0.0009
tridia	10000	0	1	1	0	2.027283-31	2	2	0.0153
trimloss	142	75	1	71	0	9.060000+00	3	2	0.0030

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
try-b	2	1	1	1	0	4.935752-25	2	2	0.0003
twirism1	343	313	0	124	-21	-9.362848-01	1	1	0.0470
twobars	2	2	1	1	0	1.508652+00	2	2	0.0004
ubh1	17997	12000	1	1	0	1.116000+00	2	2	0.0702
ubh5	19997	14000	1	1	0	1.115906+00	2	2	0.0975
vanderm1	100	199	0	1056	-21	0.000000+00	1	1	0.1839
vanderm3	100	199	0	105	-21	0.000000+00	1	1	0.0549
vanderm4	9	17	0	0	0	0.000000+00	1	1	0.0003
vardim	100	0	2	2	0	2.533251-22	3	3	0.0061
watson	31	0	2	2	0	2.514755-13	3	3	0.0012
weeds	3	0	0	0	0	9.205435+03	1	1	0.0001
womflet	3	3	1	1	0	6.050000+00	2	2	0.0004
woods	10000	0	2	2	0	1.193753-12	3	3	0.0317
yao	2002	2003	1	4425	0	1.977046+02	2	2	1.0153
yfit	3	0	1	1	0	6.753437-13	2	2	0.0006
yfitu	3	0	1	1	0	6.759538-13	2	2	0.0003
zangwil2	2	0	0	0	0	-1.819999+01	1	1	0.0002
zangwil3	3	3	1	1	0	0.000000+00	2	2	0.0003
zecevic2	2	4	1	1	0	-4.125000+00	2	2	0.0004
zecevic3	2	4	0	0	0	9.730944+01	1	1	0.0001
zecevic4	2	4	1	1	0	7.557507+00	2	2	0.0004
zigzag	64	50	3	233	0	3.161734+00	5	3	0.0069
zy2	3	2	1	1	0	2.000000+00	2	2	0.0002

B.3. With Ipopt as QP solver

Table B.3. Table of results on the CUTE test set for RestartSQP using Ipopt as QP subsolver

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
3pk	30	0	6	76	0	1.720118e+00	7	7	0.0149
aircrfta	5	5	2	13	0	0.000000e+00	3	3	0.0034
aircrftb	8	3	20	452	0	1.364822e-23	28	12	0.0662
airport	84	42	12	191	0	4.795270e+04	13	13	0.0701
aljazzaf	3	1	20	311	0	7.500499e+01	24	18	0.0401
allinit	4	3	13	213	0	1.670596e+01	16	9	0.0310
allinitc	4	4	16	345	0	3.048876e+01	18	16	0.0455
allinitu	4	0	11	120	0	5.744384e+00	12	8	0.0151
alsotame	2	3	4	32	0	8.208499e-02	5	5	0.0056
arglina	100	0	1	17	0	1.000000e+02	2	2	0.0197
arglinb	10	0	1	20	0	4.634146e+00	2	2	0.0038
arglinc	8	0	1	20	0	6.135135e+00	2	2	0.0037
argtrig	100	100	3	18	0	0.000000e+00	4	4	0.0610
artif	5000	5000	22	1226	-8	0.000000e+00	36	12	6.1030
arwhead	5000	0	6	120	0	-2.664535e-15	7	7	0.4707
aug2d	20192	9996	4	166	0	1.687411e+06	5	5	11.2424
aug2dc	20200	10194	4	199	0	1.818392e+06	5	5	13.9551
aug2dcqp	20200	10194	5	354	0	6.498179e+06	6	6	16.3920
aug2dqp	20192	10194	5	370	0	6.237011e+06	6	6	21.0607
aug3d	3873	1000	1	8	0	5.540677e+02	2	2	0.0350
aug3dc	3873	1000	1	8	0	7.712624e+02	2	2	0.0312
aug3dcqp	3873	1000	1	26	0	9.933621e+02	2	2	0.0867
aug3dqp	3873	1000	1	30	0	6.752376e+02	2	2	0.1217
avgasa	8	10	1	14	0	-4.412171e+00	2	2	0.0029
avgasb	8	10	1	16	0	-4.483219e+00	2	2	0.0032
avion2	49	15	1344	41673	0	9.468012e+07	1671	673	6.8619

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
bard	3	0	11	118	0	8.214877e-03	12	9	0.0222
batch	46	73	7	633	0	2.591803e+05	8	8	0.1356
bdexp	5000	0	10	72	0	1.963835e-03	11	11	0.1452
bdqrtic	1000	0	9	129	0	3.983817e+03	10	10	0.1494
bdvalue	5000	5000	0	0	0	0.000000e+00	1	1	0.0010
beale	2	0	17	151	0	6.721860e-14	18	12	0.0224
bigbank	2230	1112	20	776	0	-4.205696e+06	21	21	1.5900
biggs3	6	3	10	99	0	3.115215e-14	13	9	0.0143
biggs5	6	1	54	705	0	8.406381e-17	65	28	0.0871
biggs6	6	0	93	992	0	6.995473e-11	94	58	0.1162
biggsb1	1000	999	1	23	0	1.500000e-02	2	2	0.0242
biggsc4	4	7	1	32	0	-2.449999e+01	2	2	0.0052
blockqp1	2005	1001	1	12	0	-9.964999e+02	2	2	0.0235
blockqp2	2005	1001	1	13	0	-9.961011e+02	2	2	0.0258
blockqp3	2005	1001	1	23	0	-4.974999e+02	2	2	0.0368
blockqp4	2005	1001	1	14	0	-4.980982e+02	2	2	0.0265
blockqp5	2005	1001	1	101	0	-4.974999e+02	2	2	0.2166
bloweya	2002	1002	1	30	0	-1.852786e-02	2	2	0.0369
bloweyb	2002	1002	1	18	0	-1.702787e-02	2	2	0.0250
bloweyc	2002	1002	1	33	0	-1.304396e-02	2	2	0.0401
booth	2	2	1	8	0	0.000000e+00	2	2	0.0020
box2	3	1	8	68	0	1.656702e-20	10	6	0.0098
box3	3	0	7	46	0	1.011994e-15	8	8	0.0067
bqp1var	1	1	1	7	0	1.229571e-13	2	2	0.0019
bqpgabim	50	4	1	17	0	-3.790343e-05	2	2	0.0032
bqpgasim	50	0	1	20	0	-5.519813e-05	2	2	0.0034
brainpc0	6905	6900	98	4888	0	1.499638e-03	194	48	70.3225
brainpc1	6905	6900	16	910	0	9.867441e-10	31	8	11.8917
brainpc2	13805	13800	15	986	0	4.110379e-11	28	8	33.2319
brainpc3	6905	6900	73	3857	0	2.736970e-07	140	34	54.0099

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
brainpc4	6905	6900	105	5224	0	1.357826e-06	209	51	74.4317
brainpc5	6905	6900	79	4244	0	1.497035e-06	156	38	58.9591
brainpc6	6905	6900	57	3187	0	1.639157e-07	111	26	42.9293
brainpc7	6905	6900	57	2994	0	1.435337e-07	110	26	49.1765
brainpc9	6905	6900	200	10492	0	1.023217e-06	397	98	148.7290
bratu1d	1001	0	8	153	0	-8.518927e+00	9	9	0.0826
bratu2d	4900	4900	2	12	0	0.000000e+00	3	3	0.3646
bratu2dt	4900	4900	5	31	0	0.000000e+00	6	6	0.7583
bratu3d	3375	3375	3	19	0	0.000000e+00	4	4	4.5546
britgas	450	360	30	12042	-8	2.619208e+01	50	14	11.8628
brkmcc	2	0	2	13	0	1.690426e-01	3	3	0.0027
brownal	10	0	6	42	0	7.976678e-11	7	7	0.0081
brownbs	2	0	47	384	0	0.000000e+00	48	35	0.0575
brownden	4	0	8	59	0	8.582220e+04	9	9	0.0104
broydn3d	10000	10000	4	26	0	0.000000e+00	5	5	0.2747
broydn7d	1000	0	19	715	0	4.820121e+02	20	12	0.4716
broydnbd	5000	5000	5	33	0	0.000000e+00	6	6	0.2975
brybnd	5000	0	7	92	0	1.294855e-13	8	8	0.6104
bt1	2	1	8	149	0	-9.999999e-01	11	7	0.0238
bt10	2	2	6	42	0	-1.000000e+00	7	7	0.0073
bt11	5	3	7	51	0	8.248917e-01	8	8	0.0088
bt12	5	3	3	23	0	6.188118e+00	4	4	0.0043
bt13	5	2	107	2185	0	1.225686e-13	200	55	0.3229
bt2	3	1	11	84	0	3.256821e-02	12	12	0.0159
bt3	5	3	2	34	0	4.093023e+00	3	3	0.0059
bt4	3	2	12	265	0	-4.551055e+01	23	7	0.0383
bt5	3	2	38	783	0	9.617151e+02	72	21	0.1097
bt6	5	2	9	76	0	2.770447e-01	12	9	0.0121
bt7	5	3	22	437	0	3.603797e+02	34	14	0.0637
bt8	5	2	9	102	0	1.000003e+00	10	10	0.0165

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
bt9	4	2	12	159	0	-1.000000e+00	18	10	0.0234
byrdsphr	3	2	89	1539	0	-4.683300e+00	178	44	0.2184
camel6	2	2	8	113	0	-1.031628e+00	9	6	0.0207
cantilvr	5	1	3	84	-30	2.243652e+00	7	3	0.0164
catena	32	11	28	5504	0	-2.307774e+04	48	15	0.8704
catenary	496	166	286	16225	0	-3.484031e+05	404	149	5.6402
cb2	3	3	6	52	0	1.952224e+00	7	7	0.0088
cb3	3	3	5	38	0	1.999999e+00	6	6	0.0067
cbratu2d	882	882	1	6	0	0.000000e+00	2	2	0.0277
cbratu3d	1024	1024	1	6	0	0.000000e+00	2	2	0.1028
chaconn1	3	3	4	30	0	1.952224e+00	5	5	0.0055
chaconn2	3	3	4	28	0	1.999999e+00	5	5	0.0053
chainwoo	1000	0	64	1239	0	4.792811e+01	65	41	0.5813
chandheq	100	100	9	57	0	0.000000e+00	10	10	0.2097
chebyqad	50	0	74	1847	0	5.386315e-03	75	34	1.1000
chemrctb	1000	1000	3000	134928	-2	0.000000e+00	4666	1494	177.4651
chenhark	1000	0	1	23	0	-1.999999e+00	2	2	0.0104
chnrosnb	50	0	55	414	0	1.297084e-14	56	42	0.0685
cliff	2	0	27	195	0	1.997866e-01	28	28	0.0307
clnlbeam	1499	1000	3	496	0	3.448762e+02	4	4	0.6508
clplatea	4970	0	6	50	0	-1.259209e-02	7	6	0.2525
clplateb	4970	0	11	92	0	-6.988222e+00	12	10	0.4384
clplatec	4970	0	1	6	0	-5.020724e-03	2	2	0.0308
cluster	2	2	7	44	0	0.000000e+00	8	8	0.0078
concon	15	11	8	195	0	-6.230795e+03	12	9	0.0318
congigmz	3	5	5	85	0	2.799999e+01	6	6	0.0136
coolhans	9	9	20	637	0	0.000000e+00	27	16	0.0958
core1	65	115	157	13355	-4	9.390046e+01	276	54	3.7078
core2	157	134	31	6230	-24	7.243439e+01	50	16	6.1937
corkscrw	8997	7000	20	2215	0	9.068782e+01	31	11	18.6022

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
coshfun	61	20	394	16837	0	-7.732618e-01	784	193	2.7488
cosine	10000	0	29	848	0	-9.999000e+03	30	13	2.9260
cragglvy	5000	0	14	195	0	1.688215e+03	15	15	0.7500
cresc100	6	200	1720	58952	0	5.676028e-01	3435	856	20.4005
cresc4	6	8	452	20277	0	8.718975e-01	894	225	2.6985
cresc50	6	100	3000	88685	-2	5.959294e-01	5995	1497	22.3916
csfi1	5	4	20	473	0	-4.907520e+01	32	12	0.0707
csfi2	5	4	42	1179	0	5.501760e+01	72	24	0.1771
cube	2	0	38	277	0	4.115625e-17	39	28	0.0429
curly10	10000	0	13	420	0	-1.003162e+06	14	9	4.9596
curly20	10000	0	15	620	0	-1.003162e+06	16	10	17.0756
curly30	10000	0	17	698	0	-1.003162e+06	18	11	29.2805
cvxbqp1	10000	0	1	12	0	2.250225e+06	2	2	0.5577
cvxqp1	1000	500	1	701	0	1.087511e+06	2	2	4.5382
dallass	46	31	17	412	0	-3.239322e+04	20	16	0.0674
deconvb	51	0	34	667	0	3.142623e-03	35	16	0.1235
deconvc	51	1	37	1407	0	3.203098e-03	72	19	0.2674
deconvu	51	0	1303	20885	0	4.743366e-10	1304	647	4.4202
degenlpa	20	14	1	185	0	3.060349e+00	2	2	0.0242
degenlpb	20	15	1	238	0	-3.073124e+01	2	2	0.0315
demymalo	3	3	6	56	0	-3.000000e+00	8	6	0.0081
denschna	2	0	5	30	0	2.213909e-12	6	6	0.0049
denschnb	2	0	9	63	0	3.721375e-17	10	7	0.0084
denschnc	2	0	10	61	0	2.177679e-20	11	11	0.0131
denschnd	3	0	35	299	0	1.742452e-08	36	28	0.0433
denschne	3	0	9	62	0	4.066134e-13	10	10	0.0099
denschnf	2	0	6	36	0	6.513246e-22	7	7	0.0092
dipigri	7	4	9	141	0	6.806300e+02	15	6	0.0186
disc2	28	23	24	572	0	1.562500e+00	38	15	0.0915
discs	36	69	3000	114691	-2	1.569549e+01	5992	1495	27.3094

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
dittert	327	264	59	2862	0	-1.997596e+00	118	28	7.7971
dixchlng	10	5	10	246	0	2.471897e+03	12	11	0.0330
dixchlnv	100	50	17	3450	0	3.018525e-22	18	18	2.0746
dixmaana	3000	0	8	112	0	1.000000e+00	9	6	0.1260
dixmaanb	3000	0	30	1448	0	1.000000e+00	31	17	2.2565
dixmaanc	3000	0	23	1283	0	1.000000e+00	24	13	2.1467
dixmaand	3000	0	16	1375	0	1.000000e+00	17	12	2.3326
dixmaane	3000	0	12	283	0	1.000000e+00	13	8	0.2074
dixmaanf	3000	0	37	2032	0	1.000000e+00	38	20	3.0018
dixmaang	3000	0	41	2011	0	1.000000e+00	42	23	2.9795
dixmaanh	3000	0	44	2437	0	1.000000e+00	45	24	3.8908
dixmaani	3000	0	11	202	0	1.000000e+00	12	8	0.1710
dixmaanj	3000	0	49	2155	0	1.000000e+00	50	27	3.3199
dixmaank	3000	0	40	2103	0	1.000000e+00	41	22	3.0695
dixmaanl	3000	0	52	2392	0	1.000000e+00	53	28	3.4708
dixon3dq	10	0	1	8	0	1.953219e-27	2	2	0.0033
djtl	2	0	29	229	0	-8.951544e+03	30	14	0.0388
dnieper	61	24	3	110	0	1.874401e+04	4	4	0.0225
dqdrtic	5000	0	1	27	0	7.533621e-29	2	2	0.0976
dqrtic	5000	0	38	886	0	2.110693e-05	39	39	0.9708
dtoc1l	14985	9990	5	31	0	1.253381e+02	6	6	0.6390
dtoc1na	1485	990	5	37	0	1.270202e+01	7	6	0.2119
dtoc1nb	1485	990	6	73	0	1.593777e+01	11	6	0.3085
dtoc1nc	1485	990	45	1069	0	2.496981e+01	90	23	3.4497
dtoc1nd	735	490	94	2700	0	1.277580e+01	187	48	4.1056
dtoc2	5994	3996	7	94	0	5.086762e-01	9	7	0.4609
dtoc3	14997	9998	3	128	0	2.352624e+02	4	4	1.1097
dtoc4	14997	9998	3	27	0	2.868538e+00	4	4	0.2915
dtoc5	9998	4999	3	23	0	1.535102e+00	4	4	0.1322
dtoc6	10000	5000	11	181	0	1.348506e+05	12	12	0.8916

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
dual1	85	1	1	22	0	3.501296e-02	2	2	0.0105
dual2	96	1	1	17	0	3.373367e-02	2	2	0.0104
dual3	111	1	1	23	0	1.357558e-01	2	2	0.0175
dual4	75	1	1	18	0	7.460906e-01	2	2	0.0080
dualc1	9	215	1	189	0	6.155251e+03	2	2	0.0871
dualc2	7	229	1	123	0	3.551306e+03	2	2	0.0459
dualc5	8	278	1	66	0	4.272325e+02	2	2	0.0316
dualc8	8	503	1	145	0	1.830936e+04	2	2	0.0962
edensch	2000	0	7	93	0	1.200328e+04	8	8	0.1349
eg1	3	0	7	48	0	-1.429306e+00	8	7	0.0073
eg2	1000	0	3	19	0	-9.989473e+02	4	4	0.0099
eg3	101	200	25	618	0	4.998986e-11	33	13	0.2195
eigena	110	0	24	916	0	2.521753e-11	25	17	0.6025
eigena2	110	55	2	112	0	3.217073e-30	3	3	0.0822
eigenaco	110	55	2	2381	0	0.000000e+00	3	3	3.4615
eigenals	110	0	20	15022	0	5.629040e-21	21	15	13.3695
eigenb	110	0	112	2214	0	1.113201e-14	113	68	1.4477
eigenb2	110	55	2	70	0	1.800000e+01	3	3	0.0443
eigenbco	110	55	379	13925	0	5.771244e-17	748	186	12.8462
eigenbls	110	0	112	2204	0	1.113201e-14	113	68	1.6470
eigencco	30	15	69	1863	0	3.109317e-15	135	34	0.3769
eigmaxa	101	101	11	506	0	-1.300000e+01	14	10	0.1369
eigmaxb	101	101	7	134	0	-9.674353e-04	8	8	0.0415
eigmaxc	22	22	5	59	0	-1.000000e+00	6	6	0.0127
eigmina	101	101	2	30	0	9.999999e-01	3	3	0.0093
eigminb	101	101	7	102	0	9.674353e-04	8	8	0.0298
eigminc	22	22	5	80	0	9.999999e-01	6	6	0.0156
engval1	5000	0	7	103	0	5.548668e+03	8	8	0.4301
engval2	3	0	18	165	0	2.822411e-25	19	15	0.0244
errinros	50	0	51	462	0	3.990415e+01	52	36	0.0779

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
expfit	2	0	12	121	0	2.405105e-01	13	6	0.0173
expfita	5	22	12	130	0	1.136611e-03	13	13	0.0235
expfitb	5	102	14	274	0	5.019365e-03	15	15	0.0697
expfitc	5	502	14	452	0	2.330257e-02	15	15	0.2616
explin	120	0	12	131	0	-7.237562e+05	13	13	0.0250
explin2	120	0	11	107	0	-7.244591e+05	12	12	0.0193
expquad	120	10	5	202	0	-3.624599e+06	6	6	0.0403
extrasim	2	1	1	7	0	1.000000e+00	2	2	0.0019
extrosnb	10	0	0	0	0	0.000000e+00	1	1	0.0001
fccu	19	8	3	58	0	1.114910e+01	4	4	0.0101
fletcbv2	100	0	1	6	0	-5.140067e-01	2	2	0.0020
fletchcr	100	0	20	650	0	2.579539e-12	21	9	0.1152
fletcher	4	5	1	93	-8	4.000000e+00	2	2	0.0176
flosp2hh	691	0	0	0	-24	8.704390e+05	1	1	44.6641
flosp2hl	691	0	2	39	0	3.887054e+01	3	3	0.1821
flosp2hm	691	0	0	0	-24	8.704390e+05	1	1	39.2773
flosp2th	691	0	0	0	-24	8.704200e+05	1	1	60.1766
flosp2tl	691	0	1	20	0	1.000000e+01	2	2	0.0925
flosp2tm	691	0	0	0	-24	8.704200e+05	1	1	58.4854
fminsrf2	1024	0	43	871	0	1.000000e+00	44	22	0.9644
fminsurf	1024	0	71	1208	0	1.000000e+00	72	35	150.3860
freuroth	5000	0	10	116	0	6.081591e+05	11	8	0.3056
genhs28	10	8	1	9	0	9.271736e-01	2	2	0.0022
genhumps	5	0	134	2493	0	1.500287e-14	135	67	0.3159
genrose	500	0	1010	8129	0	1.000000e+00	1011	756	2.3379
gigomez1	3	3	3	38	0	-2.999999e+00	5	3	0.0064
gilbert	1000	1	18	443	0	4.820272e+02	20	19	0.2968
goffin	51	50	2	50	0	6.220801e-12	3	3	0.0309
gottfr	2	2	7	104	0	0.000000e+00	11	6	0.0168
gouldqp2	699	349	1	36	0	1.879984e-04	2	2	0.0199

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
gouldqp3	699	349	1	16	0	2.065155e+00	2	2	0.0100
gpp	250	498	5	190	0	1.440092e+04	8	6	0.7571
gridneta	13284	6724	1	78	0	3.049829e+02	2	2	0.5273
gridnetb	13284	6724	1	34	0	1.433232e+02	2	2	2.9143
gridnetc	7564	3844	1	90	0	1.618702e+02	2	2	0.5830
gridnetd	7565	3844	3	108	0	5.664443e+02	4	4	0.4906
gridnete	7565	3844	3	50	0	2.065546e+02	4	4	1.3380
gridnetf	7565	3844	3	144	0	2.421089e+02	4	4	1.5902
gridnetg	61	36	4	45	0	7.331702e+01	5	5	0.0104
gridneth	61	36	4	33	0	3.962626e+01	5	5	0.0086
gridneti	61	36	4	49	0	4.024746e+01	5	5	0.0118
growth	3	0	101	1018	0	1.004040e+00	102	72	0.1682
growthls	3	0	100	1101	0	1.004040e+00	101	71	0.1902
gulf	3	0	25	209	0	1.044508e-17	26	18	0.0303
hadamals	100	0	11	317	0	2.531641e+01	12	12	0.1659
hadamard	65	256	1	13	0	1.000000e+00	2	2	0.0087
hager1	10001	5001	1	16	0	8.807970e-01	2	2	0.0935
hager2	10000	5000	1	22	0	4.320822e-01	2	2	0.1309
hager3	10000	5000	1	13	0	1.409612e-01	2	2	0.1042
hager4	10000	5000	1	28	0	2.794030e+00	2	2	0.1548
haifam	85	150	31	1826	0	-4.500036e+01	60	18	0.6266
haifas	7	9	15	276	0	-4.500000e-01	28	8	0.0437
hairy	2	0	44	488	0	2.000000e+01	45	25	0.0631
haldmads	6	42	36	1296	0	1.223711e-04	65	19	0.2466
hanging	300	180	64	1683	0	-6.201760e+02	118	32	0.8663
harkerp2	100	0	4	395	0	-4.999999e-01	5	5	0.1718
hart6	6	0	13	131	0	-3.322886e+00	14	9	0.0215
hatflda	4	0	8	87	0	3.442012e-11	9	6	0.0125
hatfldb	4	1	8	125	0	5.572809e-03	9	6	0.0200
hatfldc	4	3	4	24	0	1.139153e-20	5	5	0.0049

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hatfldd	3	0	30	241	0	6.615113e-08	31	23	0.0407
hatflde	3	0	35	297	0	4.434401e-07	36	27	0.0435
hatfldf	3	3	15	356	0	0.000000e+00	27	9	0.0525
hatfldg	25	25	12	404	0	0.000000e+00	17	9	0.0721
hatfldh	4	7	1	25	0	-2.449999e+01	2	2	0.0042
heart6	6	6	88	2911	0	0.000000e+00	156	47	0.4502
heart6ls	6	0	1541	17844	0	1.263034e-18	1542	769	2.6417
heart8	8	8	38	1740	0	0.000000e+00	72	20	0.2549
heart8ls	8	0	305	4070	0	6.079206e-16	306	152	0.5587
helix	3	0	17	170	0	6.840700e-17	18	12	0.0254
hilberta	10	0	1	9	0	2.172410e-20	2	2	0.0021
hilbertb	50	0	1	7	0	4.172678e-28	2	2	0.0028
himmelba	2	2	1	8	0	0.000000e+00	2	2	0.0020
himmelbb	2	0	54	475	0	4.224972e-23	55	32	0.0692
himmelbc	2	2	6	60	0	0.000000e+00	8	6	0.0104
himmelbd	2	2	3000	42061	-2	0.000000e+00	3010	2997	6.4449
himmelbe	3	3	2	15	0	0.000000e+00	3	3	0.0031
himmelbf	4	0	7	60	0	3.185717e+02	8	8	0.0093
himmelbg	2	0	9	74	0	2.549820e-17	10	6	0.0114
himmelbh	2	0	7	49	0	-9.999999e-01	8	5	0.0080
himmelbi	100	12	15	140	0	-1.754999e+03	16	16	0.0429
himmelbj	45	16	26	763	0	-1.910344e+03	27	26	0.1403
himmelbk	24	14	4	87	0	5.181434e-02	5	5	0.0178
himmelp1	2	2	8	72	0	-6.205386e+01	9	9	0.0140
himmelp2	2	3	8	77	0	-6.205386e+01	9	9	0.0125
himmelp3	2	4	4	40	0	-5.901312e+01	5	5	0.0067
himmelp4	2	5	4	37	0	-5.901312e+01	5	5	0.0065
himmelp5	2	5	8	137	0	-5.901312e+01	9	9	0.0217
himmelp6	2	5	1	7	0	-5.901312e+01	2	2	0.0020
hong	4	1	7	80	0	1.347306e+00	8	8	0.0127

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs001	2	1	31	230	0	7.034741e-15	32	25	0.0365
hs002	2	1	8	61	0	4.941229e+00	9	8	0.0100
hs003	2	1	1	20	0	5.342625e-13	2	2	0.0034
hs004	2	2	2	15	0	2.666666e+00	3	3	0.0031
hs005	2	2	7	96	0	-1.913222e+00	8	6	0.0155
hs006	2	1	1	16	0	8.332343e-30	3	2	0.0031
hs007	2	1	16	185	0	-1.732050e+00	25	11	0.0280
hs008	2	2	5	34	0	-1.000000e+00	6	6	0.0060
hs009	2	1	4	29	0	-4.999999e-01	5	4	0.0053
hs010	2	1	8	57	0	-1.000000e+00	9	9	0.0095
hs011	2	1	5	37	0	-8.498464e+00	6	6	0.0065
hs012	2	1	9	120	0	-3.000000e+01	17	7	0.0195
hs013	2	1	16	313	-3	1.034983e+00	17	11	0.0499
hs014	2	2	5	39	0	1.393464e+00	6	6	0.0069
hs015	2	3	6	185	0	3.065000e+02	8	6	0.0293
hs016	2	4	6	110	0	2.314466e+01	7	7	0.0177
hs017	2	4	10	216	0	1.000000e+00	11	10	0.0317
hs018	2	4	6	60	0	5.000000e+00	9	7	0.0103
hs019	2	4	5	121	0	-6.961813e+03	6	6	0.0188
hs020	2	4	6	109	0	4.019872e+01	7	7	0.0174
hs021	2	3	1	10	0	-9.995999e+01	2	2	0.0023
hs022	2	2	4	30	0	9.999999e-01	5	5	0.0054
hs023	2	5	5	47	0	2.000000e+00	6	6	0.0083
hs024	2	3	2	15	0	-9.999999e-01	3	3	0.0032
hs025	3	3	0	0	0	3.283499e+01	1	1	0.0001
hs026	3	1	15	106	0	1.021028e-10	17	15	0.0174
hs027	3	1	49	950	0	3.999999e-02	95	27	0.1319
hs028	3	1	1	8	0	1.828801e-28	2	2	0.0023
hs029	3	1	9	170	0	-2.262741e+01	18	6	0.0258
hs030	3	4	1	8	0	1.000000e+00	2	2	0.0020

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs031	3	4	5	42	0	5.999999e+00	6	6	0.0075
hs032	3	2	1	22	0	1.000000e+00	2	2	0.0037
hs033	3	3	5	32	0	-4.585786e+00	6	6	0.0061
hs034	3	5	8	123	0	-8.340324e-01	15	7	0.0196
hs035	3	1	1	9	0	1.111111e-01	2	2	0.0021
hs036	3	4	2	73	0	-3.299999e+03	3	3	0.0117
hs037	3	2	4	79	0	-3.455999e+03	5	5	0.0131
hs038	4	0	52	417	0	2.143728e-20	53	39	0.0603
hs039	4	2	12	159	0	-1.000000e+00	18	10	0.0237
hs040	4	3	3	21	0	-2.500009e-01	4	4	0.0044
hs041	4	5	5	42	0	1.925925e+00	6	6	0.0077
hs042	4	2	5	37	0	1.385786e+01	6	6	0.0063
hs043	4	3	9	149	0	-4.400000e+01	17	7	0.0237
hs044	4	6	1	11	0	-1.299999e+01	2	2	0.0024
hs045	5	0	0	0	0	2.000000e+00	1	1	0.0001
hs046	5	2	33	663	0	6.332767e-08	64	20	0.1066
hs047	5	3	21	295	0	3.131472e-09	38	16	0.0457
hs048	5	2	1	8	0	5.980551e-29	2	2	0.0020
hs049	5	2	15	97	0	6.962487e-09	16	16	0.0165
hs050	5	3	8	85	0	1.078143e-19	9	9	0.0140
hs051	5	3	1	7	0	1.222734e-29	2	2	0.0019
hs052	5	3	1	11	0	5.326647e+00	2	2	0.0025
hs053	5	3	1	10	0	4.093023e+00	2	2	0.0022
hs054	6	1	1	8	0	1.928571e-01	2	2	0.0020
hs055	6	6	1	11	0	6.666666e+00	2	2	0.0025
hs056	7	4	85	1715	0	-3.456000e+00	165	45	0.2530
hs057	2	3	2	20	0	3.064761e-02	3	3	0.0040
hs059	2	3	9	95	0	-6.749505e+00	10	9	0.0161
hs060	3	1	6	40	0	3.256820e-02	7	7	0.0074
hs061	3	2	6	65	0	-1.436461e+02	8	7	0.0111

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs062	3	1	6	87	0	-2.627251e+04	7	6	0.0139
hs063	3	2	3000	45017	-2	9.680000e+02	3002	3001	6.8756
hs064	3	1	15	263	0	6.299842e+03	20	16	0.0406
hs065	3	4	5	77	0	9.535288e-01	7	6	0.0118
hs066	3	5	7	120	0	5.181632e-01	14	5	0.0197
hs067	10	21	34	659	0	-1.162026e+03	65	22	0.1239
hs070	4	1	63	833	0	9.401973e-03	64	34	0.1260
hs071	4	2	5	46	0	1.701401e+01	6	6	0.0079
hs072	4	6	9	227	0	7.276788e+02	11	10	0.0340
hs073	4	3	2	43	0	2.989437e+01	3	3	0.0078
hs074	4	4	10	168	0	5.126498e+03	11	11	0.0270
hs075	4	4	10	220	0	5.174412e+03	11	11	0.0347
hs076	4	3	1	9	0	-4.681818e+00	2	2	0.0021
hs077	5	2	10	88	0	2.415051e-01	14	10	0.0147
hs078	5	3	4	29	0	-2.919700e+00	5	5	0.0054
hs079	5	3	4	26	0	7.877682e-02	5	5	0.0052
hs080	5	3	7	54	0	5.394984e-02	8	8	0.0091
hs081	5	3	22	496	0	5.394967e-02	40	14	0.0751
hs083	5	3	4	81	0	-3.066553e+04	5	5	0.0136
hs084	5	3	4	119	0	-5.280335e+06	5	5	0.0180
hs085	5	48	18	745	0	-1.905155e+00	31	13	0.1606
hs086	5	10	3	56	0	-3.234867e+01	4	4	0.0104
hs087	11	6	12	310	0	8.827597e+03	15	12	0.0496
hs088	2	1	16	229	0	1.362656e+00	19	15	0.0359
hs089	3	1	22	366	0	1.362656e+00	30	17	0.0558
hs090	4	1	37	748	0	1.362656e+00	56	17	0.1120
hs091	5	1	68	1459	0	1.362656e+00	104	32	0.2228
hs092	6	1	82	1838	0	1.362656e+00	139	37	0.2816
hs093	6	2	1	195	-8	3.869985e+01	2	2	0.0290
hs095	6	4	2	30	0	1.561952e-02	3	3	0.0049

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs096	6	4	2	26	0	1.561952e-02	3	3	0.0046
hs097	6	4	6	88	0	4.071246e+00	7	7	0.0138
hs098	6	4	6	98	0	4.071246e+00	7	7	0.0149
hs099	23	18	16	1054	0	-8.310798e+08	17	17	0.1685
hs100	7	4	9	141	0	6.806300e+02	15	6	0.0225
hs100lnp	7	2	11	142	0	6.806300e+02	17	9	0.0224
hs100mod	7	4	8	140	0	6.787547e+02	15	5	0.0220
hs101	7	6	21	958	0	1.809764e+03	30	14	0.1446
hs102	7	6	20	702	0	9.118805e+02	30	14	0.1061
hs103	7	6	14	619	0	5.436679e+02	21	11	0.0922
hs104	8	6	12	233	0	3.951163e+00	22	9	0.0371
hs105	8	9	21	304	0	1.136360e+03	22	15	0.0614
hs106	8	14	1076	30014	0	7.049248e+03	2145	539	4.7253
hs107	9	14	7	264	0	5.055011e+03	8	8	0.0451
hs108	9	14	25	785	0	-8.660254e-01	46	12	0.1294
hs109	9	10	12	354	0	5.326851e+03	14	13	0.0639
hs110	10	0	4	24	0	-4.577846e+01	5	4	0.0048
hs111	10	3	68	1928	0	-4.776109e+01	134	35	0.3127
hs111lnp	10	3	68	1928	0	-4.776109e+01	134	35	0.2796
hs112	10	3	11	165	0	-4.776109e+01	12	12	0.0243
hs113	10	8	5	44	0	2.430620e+01	6	6	0.0080
hs114	10	11	27	604	0	-1.768806e+03	49	19	0.1007
hs116	13	28	528	18125	0	9.758750e+01	1005	264	3.1040
hs117	15	5	5	75	0	3.234867e+01	6	6	0.0120
hs118	15	17	2	25	0	6.648204e+02	3	3	0.0074
hs119	16	8	7	164	0	2.448996e+02	8	8	0.0254
hs21mod	7	7	1	23	0	-9.595999e+01	2	2	0.0041
hs268	5	5	1	31	0	-3.922195e-12	2	2	0.0049
hs35mod	3	2	1	22	0	2.500000e-01	2	2	0.0038
hs3mod	2	1	1	23	0	4.008242e-13	2	2	0.0037

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
hs44new	4	6	1	16	0	-1.499999e+01	2	2	0.0032
hs99exp	31	21	72	4452	0	-1.008062e+09	131	47	0.7257
hubfit	2	1	1	10	0	1.689349e-02	2	2	0.0022
hues-mod	10000	2	13	768	0	3.482448e+07	14	14	2.1688
huestis	10000	2	11	878	-8	8.358408e+10	12	12	2.4204
humps	2	0	197	2699	0	6.332003e-11	198	95	0.3523
hvycrash	202	150	29	1683	0	-9.204833e-11	32	30	0.5050
hypcir	2	2	6	64	0	0.000000e+00	8	6	0.0109
integreq	100	100	2	13	0	0.000000e+00	3	3	0.0402
jensmp	2	0	9	52	0	1.243621e+02	10	10	0.0122
kissing	127	903	124	4851	0	1.000000e+00	245	55	12.0279
kiwcresc	3	2	12	169	0	-6.776327e-08	20	8	0.0255
kowosb	4	0	17	158	0	3.075056e-04	18	9	0.0260
ksip	20	1001	1	32	0	5.757979e-01	2	2	0.0870
lakes	90	78	10	4163	-8	7.168891e+11	12	10	1.1507
launch	25	29	60	3459	-3	8.120209e+00	97	21	0.6287
lch	600	1	33	1718	0	-4.287718e+00	55	19	0.8350
liarwhd	10000	0	12	239	0	9.128279e-24	13	13	2.1534
linspanh	97	33	1	10	0	-7.700000e+01	2	2	0.0050
liswet1	10002	10000	1	227	0	3.612064e+01	2	2	2.1139
liswet10	10002	10000	1	426	0	4.948391e+01	2	2	4.0561
liswet11	10002	10000	1	211	0	4.951524e+01	2	2	2.1925
liswet12	10002	10000	1	4818	0	-3.314380e+03	2	2	45.1656
liswet2	10002	10000	1	70	0	2.499996e+01	2	2	0.7235
liswet3	10002	10000	1	28	0	2.499980e+01	2	2	0.2933
liswet4	10002	10000	1	29	0	2.499980e+01	2	2	0.2820
liswet5	10002	10000	1	30	0	2.499981e+01	2	2	0.3086
liswet6	10002	10000	1	32	0	2.499986e+01	2	2	0.3203
liswet7	10002	10000	1	1141	0	4.987922e+02	2	2	10.3169
liswet8	10002	10000	1	1900	0	7.144874e+02	2	2	17.4555

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
liswet9	10002	10000	1	5024	0	1.963305e+03	2	2	45.9733
lminsurf	15625	496	52	973	0	9.000000e+00	53	28	41.7000
loadbal	31	31	7	90	0	4.528510e-01	8	8	0.0174
loghairy	2	0	249	2611	0	1.823215e-01	250	142	0.3478
logros	2	0	49	472	0	0.000000e+00	50	32	0.0700
lootsma	3	3	5	32	0	1.414213e+00	6	6	0.0060
lotschd	12	7	2	63	0	2.398415e+03	3	3	0.0105
lsnnodoc	5	4	6	89	0	1.231124e+02	7	7	0.0141
lsqfit	2	1	1	10	0	3.378698e-02	2	2	0.0022
madsen	3	6	11	184	0	6.164324e-01	17	9	0.0286
madsschj	81	158	39	871	0	-7.972837e+02	65	20	2.5599
makela1	3	2	7	91	0	-1.414213e+00	11	5	0.0143
makela2	3	3	4	39	0	7.200000e+00	5	5	0.0064
makela3	21	20	102	3122	0	6.076394e-10	192	46	0.4877
makela4	21	40	2	23	0	5.018208e-12	3	3	0.0053
mancino	100	0	5	59	0	1.677174e-21	6	6	0.0396
manne	1094	730	7	1382	0	-9.745725e-01	11	4	1.8686
maratos	2	1	8	107	0	-1.000000e+00	15	5	0.0165
maratosb	2	0	9	172	0	-1.000000e+00	10	7	0.0236
matrix2	6	2	10	111	0	9.536747e-07	11	11	0.0169
maxlika	8	8	21	302	0	1.136360e+03	22	15	0.0602
mccormck	50000	0	7	76	0	-4.566161e+04	8	6	1.6678
mconcon	15	16	8	186	0	-6.230795e+03	13	9	0.0340
mdhole	2	0	2	16	0	1.254430e-13	3	3	0.0048
methanb8	31	0	38	610	0	6.372356e-17	39	19	0.1031
methanl8	31	0	104	2178	0	1.438438e-20	105	54	0.3821
mexhat	2	0	9	97	0	-4.009999e-02	10	7	0.0146
meyer3	3	0	16	224	-24	6.211027e+04	17	12	2.3475
mifflin1	3	2	10	146	0	-1.000000e+00	18	6	0.0228
mifflin2	3	2	13	216	0	-1.000000e+00	24	9	0.0319

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
minc44	311	262	18	1080	0	2.573028e-03	32	11	2.6556
minmaxbd	5	20	32	757	0	1.157064e+02	52	17	0.1231
minmaxrb	3	4	1	23	0	5.037081e-13	3	2	0.0041
minsurf	64	32	1	8	0	1.000000e+00	2	2	0.0027
mistake	9	13	25	741	0	-1.000000e+00	49	12	0.1172
model	1831	339	1	253	0	5.742163e+03	2	2	0.0841
morebv	5002	2	0	0	0	1.039542e-11	1	1	0.0018
mosarqp1	2500	700	1	16	0	-9.528754e+02	2	2	0.0311
mosarqp2	900	600	1	17	0	-1.597482e+03	2	2	0.0273
mwright	5	3	9	103	0	2.497880e+01	11	8	0.0163
nasty	2	0	1	7	0	3.260214e-30	2	2	0.0018
ncvxbqp1	10000	0	1	260	0	-1.985543e+10	2	2	19.3703
ncvxbqp2	10000	0	1	1255	0	-1.331266e+10	2	2	114.2473
ncvxqp1	1000	500	1	6135	0	-7.150698e+07	2	2	93.3200
ncvxqp2	1000	500	0	3769	-32	-2.812500e+05	1	1	66.8926
ncvxqp3	1000	500	1	4658	0	-3.028511e+07	2	2	84.3789
ncvxqp4	1000	250	1	4984	0	-9.397376e+07	2	2	42.8570
ncvxqp5	1000	250	1	4742	0	-6.625286e+07	2	2	41.3188
ncvxqp6	1000	250	1	3720	0	-3.415570e+07	2	2	33.4697
ncvxqp7	1000	750	0	5136	-32	-4.924687e+05	1	1	127.3195
ncvxqp8	1000	750	0	6546	-32	-2.812500e+05	1	1	162.3879
ncvxqp9	1000	750	1	5174	0	-2.151924e+07	2	2	132.2894
ngone	100	1273	13	1225	0	-6.332838e-01	25	7	2.7639
noncvxun	1000	0	51	25210	0	2.318491e+03	52	28	8.2986
nondia	9999	0	5	81	0	2.306591e-25	6	6	0.5829
nondquar	10000	0	19	163	0	4.139821e-10	20	20	0.8096
nonmsqrt	9	0	466	8786	-3	7.518004e-01	467	361	2.1599
nonscomp	10000	0	8	129	0	6.785647e-10	9	9	0.5802
nuffield_continuum	2	1	3	20	0	-2.549414e+00	4	4	0.0041
obstclal	96	0	1	14	0	1.397897e+00	2	2	0.0033

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
obstclbl	96	0	1	13	0	2.875038e+00	2	2	0.0028
obstclbu	96	0	1	13	0	2.875038e+00	2	2	0.0029
odfits	10	6	9	141	0	-2.380026e+03	10	10	0.0220
oet1	3	1002	1	38	0	5.382431e-01	2	2	0.0345
oet2	3	1002	5	155	0	8.715963e-02	7	6	0.1573
oet3	4	1002	1	19	0	4.505052e-03	2	2	0.0209
oet7	7	1002	1008	53353	0	4.445574e-05	2012	502	55.7626
optcdeg2	1199	800	3	221	0	2.295734e+02	4	4	0.1538
optcdeg3	1199	800	3	157	0	4.614566e+01	4	4	0.1119
optcntrl	32	21	2	71	0	5.500000e+02	3	3	0.0121
optctrl3	122	81	0	302	-8	1.560050e+06	1	1	0.0652
optctrl6	122	81	0	302	-8	1.560050e+06	1	1	0.0587
optmass	66	55	8	898	0	-1.895425e-01	15	5	0.1943
optprloc	30	30	6	226	0	-1.641977e+01	13	5	0.0416
orthrdm2	4003	2000	5	40	0	1.555328e+02	7	6	0.1336
orthrds2	203	100	3000	168339	-2	8.444699e+02	5986	1495	38.9906
orthrega	517	256	89	3936	0	1.414055e+03	173	46	2.2090
orthregb	27	6	1	7	0	0.000000e+00	2	2	0.0021
orthrege	36	20	3000	58712	-2	6.076161e-01	5997	1497	9.2466
orthrgdm	10003	5000	2	1766	-24	6.010870e+03	4	3	95.3252
orthrgds	10003	5000	1	547	-24	1.663221e+03	3	2	87.2181
osbornea	5	0	3000	71141	-2	4.696251e-02	3001	1500	13.3056
osborneb	11	0	18	164	0	4.013773e-02	19	13	0.0231
oslbqp	8	8	1	22	0	6.250000e+00	2	2	0.0032
palmer1	4	0	32	397	0	1.175460e+04	37	19	0.0521
palmer1a	6	0	48	776	0	8.988362e-02	49	28	0.1141
palmer1b	4	0	30	394	0	3.447354e+00	31	21	0.0603
palmer1c	8	0	2	67	-24	1.254715e+04	3	3	2.7411
palmer1d	7	0	5	129	0	6.526825e-01	6	6	0.0266
palmer1e	8	0	104	2276	0	8.352682e-04	105	73	0.5322

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
palmer2	4	0	38	506	-3	6.396239e+03	39	12	0.0695
palmer2a	6	0	73	791	0	1.716073e-02	74	52	0.1173
palmer2b	4	0	21	233	0	6.233946e-01	22	16	0.0340
palmer2c	8	0	4	107	0	1.442139e-02	5	5	0.0272
palmer2e	8	0	13	332	-24	2.956358e-02	14	6	3.5179
palmer3	4	0	33	397	0	2.265958e+03	34	17	0.0520
palmer3a	6	0	83	891	0	2.043142e-02	84	60	0.1249
palmer3b	4	0	33	393	0	4.227647e+00	34	21	0.0516
palmer3c	8	0	4	89	0	1.953763e-02	5	5	0.0174
palmer3e	8	0	453	5441	0	5.074084e-05	454	326	0.8333
palmer4	4	0	32	389	0	2.285383e+03	33	17	0.0527
palmer4a	6	0	57	615	0	4.060613e-02	58	40	0.0787
palmer4b	4	0	33	384	0	6.835138e+00	34	21	0.0463
palmer4c	8	0	5	114	0	5.031069e-02	6	6	0.0222
palmer4e	8	0	41	789	0	1.480042e-04	42	28	0.1427
palmer5a	8	0	3000	45081	-2	4.694674e-02	3004	1591	8.1198
palmer5b	9	0	859	11527	0	9.752492e-03	862	435	1.7581
palmer5c	6	0	3	31	0	2.128086e+00	4	4	0.0045
palmer5d	4	0	4	49	0	8.733939e+01	5	5	0.0066
palmer5e	8	0	3000	34594	-2	2.110970e-02	3001	2015	5.6025
palmer6a	6	0	131	1342	0	5.594883e-02	132	94	0.1706
palmer6c	8	0	5	113	0	1.638742e-02	6	6	0.0283
palmer6e	8	0	58	653	0	2.239550e-04	59	36	0.0910
palmer7a	6	0	3000	47007	-2	1.033667e+01	3001	2051	8.7401
palmer7c	8	0	7	153	0	6.019856e-01	8	8	0.0329
palmer7e	8	0	108	2255	0	1.015389e+01	109	52	0.3360
palmer8a	6	0	93	1058	0	7.400969e-02	94	56	0.1294
palmer8c	8	0	6	133	0	1.597680e-01	7	7	0.0319
palmer8e	8	0	116	2252	0	6.340517e-01	117	56	0.3426
penalty1	1000	0	47	905	0	9.686340e-03	48	44	95.2224

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
penalty2	100	0	19	116	0	9.709608e+04	20	20	0.0530
pentagon	6	15	18	282	0	1.365219e-04	19	11	0.0387
pentdi	1000	0	1	21	0	-7.499999e-01	2	2	0.0110
pfit1	3	0	365	3093	0	3.876847e-14	366	271	0.3947
pfit1ls	3	0	365	3093	0	3.876847e-14	366	271	0.3749
pfit2	3	0	233	2125	0	1.210046e-16	234	179	0.2841
pfit2ls	3	0	233	2125	0	1.210046e-16	234	179	0.2874
pfit3	3	0	166	1581	0	2.703838e-15	167	122	0.2176
pfit3ls	3	0	166	1581	0	2.703838e-15	167	122	0.2153
pfit4	3	0	105	1055	0	2.802046e-20	106	80	0.1504
pfit4ls	3	0	105	1055	0	2.802046e-20	106	80	0.1524
polak1	3	2	7	60	0	2.718281e+00	8	8	0.0084
polak2	11	2	17	347	0	5.459815e+01	26	12	0.0470
polak3	12	10	44	1344	0	5.933003e+00	78	21	0.1872
polak4	3	3	4	38	0	3.478794e-13	5	5	0.0059
polak5	3	2	51	1274	0	5.000000e+01	93	28	0.1614
polak6	5	4	42	742	0	-4.400000e+01	70	19	0.1021
porous1	4900	4900	14	397	0	0.000000e+00	18	12	13.2729
porous2	4900	4900	11	2697	0	0.000000e+00	16	8	85.4244
portfl1	12	1	1	11	0	2.048627e-02	2	2	0.0025
portfl2	12	1	1	11	0	2.968923e-02	2	2	0.0028
portfl3	12	1	1	12	0	3.274970e-02	2	2	0.0023
portfl4	12	1	1	11	0	2.630695e-02	2	2	0.0039
portfl6	12	1	1	10	0	2.579179e-02	2	2	0.0020
powell20	1000	1000	5	477	0	5.214578e+07	6	6	0.4646
powellbs	2	2	24	435	0	0.000000e+00	43	17	0.0564
powellsq	2	2	48	968	0	0.000000e+00	76	27	0.1252
power	1000	0	1	18	0	2.243323e-30	2	2	0.0184
probpenl	500	0	32	1512	0	-3.793242e+05	33	33	45.8092
prodpl0	60	29	10	184	0	6.091923e+01	13	9	0.0342

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
prodpl1	60	29	8	129	0	5.303701e+01	10	7	0.0244
pspdoc	4	1	12	225	0	2.414213e+00	13	8	0.0315
pt	2	501	1	37	0	1.783942e-01	2	2	0.0163
qpcboei1	384	348	8	1088	0	1.443386e+07	9	9	1.0171
qpcboei2	143	140	7	979	0	8.293665e+06	8	8	0.4046
qpcstair	467	356	9	1174	0	6.204391e+06	10	10	1.0354
qpnboei1	384	348	7	3098	0	8.441646e+06	8	8	4.0689
qpnboei2	143	140	8	2443	0	1.285269e+06	9	9	1.3672
qpnstair	467	356	7	1085	0	5.146033e+06	8	8	1.1208
qr3d	155	0	400	5469	0	1.056285e-17	401	201	4.4267
qr3dbd	127	0	109	1271	0	1.463707e-15	110	55	0.6970
qr3dls	155	0	400	5489	0	1.056276e-17	401	201	4.6852
qrtquad	120	10	16	525	0	-3.648088e+06	17	12	0.0948
quartc	10000	0	41	1868	0	2.620441e-05	42	42	3.9871
qudlin	12	0	1	33	0	-7.199999e+03	2	2	0.0042
reading1	10001	5000	38	3292	0	-1.602479e-01	76	17	15.5652
reading2	15003	10002	1	29	0	-1.258248e-02	2	2	0.2418
reading3	202	102	8	269	0	-1.393623e-12	16	2	0.0654
recipe	3	3	2	15	0	0.000000e+00	3	3	0.0031
res	18	14	0	0	0	0.000000e+00	1	1	0.0001
rk23	17	11	8	163	0	8.333333e-02	13	7	0.0278
robot	14	9	6	370	0	1.339073e+01	8	7	0.0510
rosenbr	2	0	28	191	0	4.260063e-22	29	22	0.0248
rosenmmx	5	4	34	593	0	-4.400000e+01	57	15	0.0781
s332	2	102	11	288	0	2.992435e+01	19	9	0.0611
s365mod	7	9	17	1119	-8	2.214991e+04	27	12	0.1485
s368	100	0	0	0	0	0.000000e+00	1	1	0.0058
sawpath	593	786	0	0	-32	1.483193e+03	1	1	0.0876
scon1dls	1000	0	286	3925	0	6.954942e-10	287	239	1.8589
scosine	10000	0	0	0	-24	8.774948e+03	1	1	32.6125

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
scurly10	10000	0	0	0	-24	7.308607e+15	1	1	124.9546
semicon1	1000	1000	6	272	-30	0.000000e+00	8	7	0.2159
semicon2	1000	1000	48	3551	0	0.000000e+00	70	29	3.5591
sim2bqp	2	0	1	22	0	2.809411e-13	2	2	0.0037
simbqp	2	1	1	23	0	5.103734e-13	2	2	0.0036
simpllpa	2	2	1	10	0	1.000000e+00	2	2	0.0034
simpllpb	2	3	1	11	0	1.100000e+00	2	2	0.0020
sineali	20	0	10	159	0	-1.899796e+03	11	5	0.0213
sineval	2	0	56	439	0	1.409214e-18	57	42	0.0730
sinquad	10000	0	10	256	-24	1.891797e-03	11	6	143.5391
sinrosnb	1000	1000	0	0	0	-9.990100e+04	1	1	0.0006
sipow1	2	10000	1	26	0	-1.000000e+00	2	2	0.1881
sipow1m	2	10000	1	27	0	-1.000000e+00	2	2	0.1897
sipow2	2	5001	1	23	0	-1.000000e+00	2	2	0.0859
sipow2m	2	5001	1	23	0	-1.000000e+00	2	2	0.0865
sipow3	4	10000	1	20	0	5.356507e-01	2	2	0.2091
sipow4	4	10000	1	21	0	2.728283e-01	2	2	0.1984
sisser	2	0	16	118	0	1.763695e-08	17	13	0.0203
smbank	117	64	14	366	-32	-5.209253e+06	15	15	0.1435
smmpsf	720	263	22	726	0	1.046985e+06	23	23	0.3994
snake	2	2	2	83	0	5.910853e-14	3	3	0.0110
sosqp1	20000	10001	1	14	0	-6.203209e-10	2	2	0.3632
sosqp2	20000	10001	1	33	0	-4.998699e+03	2	2	0.4429
spanhyd	97	33	2	1743	-32	1.809102e+04	3	3	0.7689
spiral	3	2	116	2529	0	-6.789938e-11	221	90	0.3335
sreadin3	10001	5001	1	22	0	-7.308899e-05	2	2	0.1317
srosenbr	10000	0	28	319	0	2.191610e-18	29	22	1.2812
sseblin	194	72	10	407	0	1.617060e+07	11	11	0.0995
ssebnln	194	96	6	1458	-8	1.917493e+07	7	7	0.4234
ssnlbeam	33	20	4	164	0	3.377724e+02	5	5	0.0246

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
stancmin	3	2	5	75	0	4.250000e+00	6	6	0.0138
steenbra	432	108	8	241	0	1.695767e+04	9	9	0.5841
steenbrb	468	108	2	145	-24	1.141837e+05	3	3	53.0812
steenbrc	540	126	3000	170908	-2	2.619867e+04	3923	1496	76.4851
steenbre	540	126	81	4519	-3	2.824541e+04	87	37	36.7733
steenbrf	468	108	109	24777	-33	3.521270e+02	112	55	10.3788
supersim	2	2	1	8	0	6.666666e-01	2	2	0.0022
svanberg	5000	5000	8	302	0	8.361422e+03	15	8	4.2001
swopf	83	92	4	63	0	6.786014e-02	5	5	0.0176
synthes1	7	7	4	204	0	5.171320e+00	9	5	0.0266
tame	2	1	1	8	0	0.000000e+00	2	2	0.0020
tfi2	3	10001	1	38	0	6.490420e-01	2	2	0.2907
tointqor	50	0	1	7	0	1.175472e+03	2	2	0.0026
trainf	20008	10002	3	98	0	3.103384e+00	4	4	0.9514
trainh	20008	10002	5	822	0	1.231197e+01	6	6	11.6358
tridia	10000	0	1	18	0	2.915330e-24	2	2	0.1085
trimloss	142	75	4	116	0	9.060000e+00	5	5	0.0311
try-b	2	1	6	85	0	1.000002e+00	7	7	0.0113
twirism1	343	313	24	15424	-33	-5.197412e-02	40	13	56.1140
twobars	2	2	11	160	0	1.508652e+00	20	9	0.0262
ubh1	17997	12000	8	449	0	1.116000e+00	9	9	5.0441
ubh5	19997	14000	10	499	0	1.116000e+00	11	11	7.6995
vanderm2	100	199	18	6022	-8	0.000000e+00	34	8	28.1966
vanderm3	100	199	33	11319	-8	0.000000e+00	58	15	52.3997
vanderm4	9	17	13	1067	-8	0.000000e+00	22	9	0.1679
vardim	100	0	0	0	-24	1.310583e+14	1	1	10.1184
watson	31	0	12	263	0	2.513681e-09	13	13	0.0777
weeds	3	0	37	399	0	2.587277e+00	38	26	0.0561
womflet	3	3	22	307	0	1.444300e-10	34	13	0.0414
woods	10000	0	50	649	0	2.297634e-10	51	38	3.2568

name	nVar	nConstr	iter	QP_iter	exitflag	objective	#obj	#grad	cpu_time
yao	2002	2003	2	1323	0	1.977046e+02	3	3	2.7741
yfit	3	0	46	386	0	6.669720e-13	47	39	0.0638
yfitu	3	0	46	386	0	6.669720e-13	47	39	0.0626
zangwil2	2	0	1	7	0	-1.820000e+01	2	2	0.0020
zangwil3	3	3	4	64	0	0.000000e+00	5	5	0.0132
zecevic2	2	4	1	9	0	-4.124999e+00	2	2	0.0035
zecevic3	2	4	45	865	0	9.730945e+01	89	23	0.1366
zecevic4	2	4	5	41	0	7.557507e+00	6	6	0.0099
zigzag	64	50	9	287	0	3.161734e+00	13	8	0.0618
zy2	3	2	4	28	0	2.000000e+00	5	5	0.0074

APPENDIX C

Schur Complement View of the Log-Barrier Approximation

We consider the log-barrier approximation discussed in Chapter 4.3.3 for the second-stage problem and assume that Assumption 4.5.1 holds. Let $u_0 = (z, x, \hat{\eta}, \xi_0), u_i = (y_i, s_i, \xi_i, \eta_i, \lambda_i)$ and $u = (u_0, u_1, \ldots, u_N)$, To solve the nonlinear system of equations (4.24), we aim to compute the Newton step Δu by solving the following linear system (C.1):

$$\text{(C.1)} \qquad \begin{bmatrix} \nabla_{u_0} F_0^T & \nabla_{u_1} F_0^T & \cdots & \nabla_{u_N} F_0^T \\ \nabla_{u_0} F_1^T & \nabla_{u_1} F_1^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \nabla_{u_0} F_N^T & 0 & \cdots & \nabla_{u_N} F_N^T \end{bmatrix} \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_N \end{bmatrix} = - \begin{bmatrix} F_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

In this linear system, $\nabla_{u_0} F_0^T$ represents the Jacobian of F_0 with respect to u_0, $\nabla_{u_i} F_0^T$ represents the Jacobian of F_0 with respect to u_i, and $\nabla_{u_i} F_i^T$ represents the Jacobian of F_i with respect to u_i. Assuming that $\nabla_{u_0} F_0$ is invertible, we can express Δu_0 as follows:

$$\Delta u_0 = - \left(\nabla_{u_0} F_0^T - \sum_{i=1}^{N} \nabla_{u_i} F_0^T (\nabla_{u_i} F_i)^{-T} \nabla_{u_0} F_i^T \right)^{-1} F_0$$

where $\nabla_{u_i} F_i^T$ is given by equation (4.12), and the expression for $\nabla_{u_0} F_0^T$ is:

$$\nabla_{u_0} F_0^T = \begin{bmatrix} \nabla_{zz}^2 \mathcal{L}_0 & 0 & \hat{P}^T & \nabla_z c_0^T \\ 0 & 0 & -I & 0 \\ \hat{P} & -I & 0 & 0 \\ \nabla_z c_0 & 0 & 0 & 0 \end{bmatrix}$$

Furthermore, we have:

$$\nabla_{u_i} F_0 = \nabla_{u_0} F_i^T = \begin{bmatrix} 0 & -E_i & 0 & 0 \end{bmatrix}$$

where E_i is a matrix of the form:

$$E_i = \begin{bmatrix} 0 & 0 & 0 & I_{x,i} & 0 \end{bmatrix}^T$$

Here, $I_{x,i}$ is a matrix that has the block corresponding to x_i as the identity matrix and the remaining blocks as zeros. With that, we have

$$\left(\nabla_{u_0} F_0^T - \sum_{i=1}^{N} \nabla_{u_i} F_0^T (\nabla_{u_i} F_i)^{-T} \nabla_{u_0} F_i^T \right) =$$

$$\begin{bmatrix} \nabla_{zz}^2 L_0 & 0 & \hat{P}^T & \nabla_z c_0^T \\ 0 & -\sum_{i=1}^{N} E_i^T (\nabla_{u_i} F_i)^{-T} E_i & -I & 0 \\ \hat{P} & -I & 0 & 0 \\ \nabla_z c_0 & 0 & 0 & 0 \end{bmatrix}.$$

With the given expressions, we can reorder equation (C.1) and expand Δu_0 and F_0 as follows:

$$\begin{bmatrix} \nabla_{zz}^2 L_0 & 0 & \hat{P}^T & \nabla_z c_0^T \\ 0 & -\sum_{i=1}^{N} E_i^T (\nabla_{u_i} F_i)^{-T} E_i & -I & 0 \\ \hat{P} & -I & 0 & 0 \\ \nabla_z c_0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x \\ \Delta \hat{\eta} \\ \Delta \xi_0 \end{bmatrix} = \begin{bmatrix} -\nabla_z \mathcal{L}_0 \\ -(\nabla_x \mathcal{L}_0 - \eta) \\ -(\hat{P} z - x) \\ -c_0(z) \end{bmatrix}$$

We can see that Δu_0 is the solution to the following SQP subproblem:

$$\begin{aligned}
\min \quad & \frac{1}{2}\Delta z^T \nabla^2_{zz} L_0 \Delta z + \frac{1}{2}\sum_{i=1}^{N} \Delta x_i^T (-I_{x,i}^T (\nabla_{u_i} F_i)^{-T} I_{x,i}) \Delta x_i \\
& + \nabla_z f_0(z)^T \Delta z + \sum_{i=1}^{N} (-\eta_i)^T \Delta x_i \\
\text{s.t.} \quad & c_0(z) + \nabla_z c_0(z)^T \Delta z = 0, \\
& \hat{P}_i(z + \Delta z) - (x_i + \Delta x_i) = 0, \quad \forall i \in [N]
\end{aligned}$$

From the given formulation, we can observe that the gradient $\nabla_{x_i} \hat{f}_i$ can be expressed as $-\eta_i$, and the Hessian $\nabla^2_{x_i x_i} \hat{f}_i$ of the objective function of the i-th subproblem can be expressed as $(-I_{x,i}^T (\nabla_{u_i} F_i)^{-T} I_{x,i})$.

We further note that the matrix $I_{x,i}(\nabla_{u_i} F_i)^{-T} I_{x,i}$ is a submatrix of $\nabla_{u_i} F_i^{-T}$, with both rows and columns corresponding to the variables x_i. By examining equation (4.13), we can observe that $\nabla_{x_i} u_i^T$ is essentially the submatrix of $\nabla_{u_i} F_i^{-T}$ with columns corresponding to x_i. And from the definition of u_i in this chapter, $\nabla_{x_i} \eta_i^T$ is a submatrix of $\nabla_{x_i} u_i^T$ with rows corresponding to the variable x_i. Thus, we can conclude that $\nabla_{x_i} \eta_i^T = I_{x,i}^T (\nabla_{u_i} F_i)^{-T} I_{x,i}$, which is consistent with the discussion in Lemma 4.3.1.