On the Implementation of an Interior-Point Algorithm for Nonlinear Optimization with Inexact Step Computations

Frank E. Curtis
Lehigh University

Johannes Huber
University of Basel

Olaf Schenk
University of Basel

Andreas Wachter
Northwestern University

Report: 12T-006

Noname manuscript No.
(will be inserted by the editor)

On the Implementation of an Interior-Point Algorithm for Nonlinear Optimization with Inexact Step Computations

Frank E. Curtis · Johannes Huber · Olaf Schenk · Andreas Wächter

March 27, 2012

Abstract This paper describes a practical implementation of a line-search interior-point algorithm for large-scale nonlinear optimization. It is based on the algorithm proposed by Curtis, Schenk, and Wächter [*SIAM J. Sci. Comput.*, 32 (2010), pp. 3447-3475], a method that possesses global convergence guarantees to first-order stationary points with the novel feature that inexact search direction calculations are allowed in order to save computational expense during each iteration. The implementation follows the proposed algorithm, except that additional functionality is included to avoid the explicit computation of a normal step during every iteration. It also contains further enhancements that have not been studied along with the previous theoretical analysis. The implementation has been included in the `IPOPT` software package paired with an iterative linear system solver and preconditioner provided in the `PARDISO` software. Numerical results on a large nonlinear optimization test set and two PDE-constrained optimization problems with control and state constraints are presented to illustrate that the implementation is robust and efficient for large-scale applications.

Keywords large-scale optimization, PDE-constrained optimization, interior-point methods, nonconvex programming, line search, trust regions, inexact linear system solvers, Krylov subspace methods

Frank E. Curtis
Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA.
This author was supported by National Science Foundation grant DMS-1016291.
E-mail: frank.e.curtis@lehigh.edu

Johannes Huber
Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland.
E-mail: johannes.huber@unibas.ch

Olaf Schenk
Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland.
E-mail: olaf.schenk@unibas.ch

Andreas Wächter
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA.
E-mail: andreas.waechter@northwestern.edu

Mathematics Subject Classification (2000) 49M05 · 49M15 · 49M37 · 65F10 · 65K05 · 65N22 · 90C06 · 90C26 · 90C30 · 90C51 · 90C90

1 Introduction

The techniques described in this paper are motivated by increased interests in the solution of large-scale nonlinear optimization problems. By large-scale, we refer to classes of problems for which contemporary optimization techniques, including most interior-point methods, have proved to be impractical due to large numbers of variables/constraints and significant fill-in during the factorization of derivative matrices. New computationally efficient strategies are needed if such large-scale problems are to be solved realistically in practical situations.

The main purpose of this paper is to describe a practical implementation, including enhanced algorithmic features, for the algorithm proposed and analyzed in [20]. This algorithm addresses the challenges posed in large-scale nonlinear optimization by employing iterative linear system solvers in place of direct factorization methods when solving the large-scale linear systems involved in an interior-point strategy. Moreover, computational flexibility is greatly increased as inexact search direction calculations are allowed, but controlled sufficiently so that theoretical convergence guarantees are maintained. Our experience has shown that the implementation described in this paper achieves these desirable characteristics.

A prime example of a class of problems for which our techniques may be applicable are those where the constraints involve discretized partial differential equations (PDEs) [6,7,12,30]. Typical methods for solving these types of problems generally fall into the categories of nonlinear elimination [2,5,22,34,50], reduced space [28,29,31,38,46], or full space [8,9,10,27,37] techniques. The algorithm discussed in this paper fits into the category of full-space methods, but is unique from many previously proposed approaches in its ability to attain strong theoretical convergence guarantees with great computational flexibility.

We describe our implementation in the context of the generic problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \; & f(x) \\ \text{s.t. } & c_{\mathcal{E}}(x) = 0 \\ & c_{\mathcal{I}}(x) \geq 0, \end{aligned} \tag{1.1}$$

where the objective $f : \mathbb{R}^n \to \mathbb{R}$, equality constraints $c_{\mathcal{E}} : \mathbb{R}^n \to \mathbb{R}^p$, and inequality constraints $c_{\mathcal{I}} : \mathbb{R}^n \to \mathbb{R}^q$ are assumed to be sufficiently smooth (e.g., C^2). If problem (1.1) is infeasible, however, then our algorithm is designed to return an approximate stationary point for the unconstrained problem

$$\min_{x\in\mathbb{R}^n} \; \tfrac{1}{2}\|c_{\mathcal{E}}(x)\|_2^2 + \tfrac{1}{2}\|\max\{-c_{\mathcal{I}}(x), 0\}\|_2^2 \tag{1.2}$$

as a certificate of infeasibility. Here, the "max" of vector quantities is to be understood component-wise. A solution to (1.2) that does not satisfy the constraints of problem (1.1) is known as an infeasible stationary point of the optimization problem.

While the algorithm described in this paper is similar to the one presented in [20], we present its implementation here in much more detail. In addition, there

are a number of notable differences. The primary difference is the strategy we describe for switching between two potential search computation methods. As described in §2.2, our goal in this strategy is to improve the overall efficiency of the algorithm while still relying on the theoretical convergence guarantees provided by the techniques developed in [20]. The other main differences include refinements of the implemented termination tests for the iterative linear system solver and Hessian modification strategy, as well as a new adaptive refinement strategy in the preconditioner computation.

The focus of our numerical experiments is the performance of the nonlinear optimization algorithm when an iterative solver is used for the inexact solution of the arising linear systems. We note, however, that in practice it is also extremely important to employ effective preconditioners. For example, in the context of PDE-constrained optimization, advantages can be gained by exploiting spectral properties of discrete differential operators. In our study, we obtain very encouraging results using the general-purpose preconditioner implemented in the `PARDISO` software package when the algorithm is applied to model PDE-constrained problems; see §3.4. Limitations of this approach are commented on further in §5.

Notation. All norms are considered ℓ_2 unless otherwise indicated. A vector composed of stacked subvectors is written in text as an ordered list of subvectors; i.e., by (a, b), we mean $[a^T\ b^T]^T$. Parenthesized superscripts (e.g., $x^{(i)}$) are used to indicate the component of a vector and subscripts are generally used to indicate the current iteration number in an algorithm. We often drop function dependencies once they are defined and, when applicable, apply iteration number information to the function itself; i.e., by f_k, we mean $f(x_k)$.

2 Algorithm Description

We motivate and describe our implemented algorithm in this section. The method is based on the series of inexact SQP, Newton, and interior-point algorithms that have been proposed and analyzed in [13, 14, 19, 20], though the majority relates to the latest enhancements in [20]. We begin by describing the basic interior-point framework of the approach, and then discuss at length the major computational component of the algorithm: namely, the search direction calculation. Further specifications and details of our software implementation are provided in §3 and §4.

It is important to note that, in the following discussion, we consider *scaled* derivatives corresponding to the slack variables for the inequality constraints; see $\gamma(z;\mu)$, $A(z)$, and $W(z,\lambda;\mu)$ throughout this section. This results in *scaled* sets of equations for computing the primal-dual step; i.e., while our focus will be on linear systems for the computation of the pair (d_k, δ_k), the algorithm will follow a search direction $\widetilde{d}_k$ where the components corresponding to the slack variables are scaled as $\widetilde{d}_k^s = \Sigma_k d_k^s$. Here, Σ is a diagonal scaling matrix that depends on the values of the slack variables s. (See [15] for a similar use of a scaled space for the slack variables.) The analysis in [20] uses $\Sigma = S := \mathrm{diag}(s)$, but it is easy to see that the convergence proofs in that paper also hold for $\Sigma^{(i,i)} = \min\{1, s^{(i)}\}$. This latter option is used in our implementation as we have experienced it to work better in practice. Following the primal-dual strategy in [49], our implementation

also maintains multipliers for the simple bounds on the slack variables. However, for ease of exposition, we suppress discussion of these multipliers until §3.2.

2.1 Interior-Point Framework

The framework of the algorithm is a classical interior-point or barrier strategy. Let $z = (x, s)$ be the primal variables, where $s \in \mathbb{R}^q$ is a vector of slack variables, and let

$$\varphi(z;\mu) := f(x) - \mu \sum_{i=1}^{q} \ln s^{(i)} \quad \text{and} \quad c(z) := \begin{bmatrix} c_{\mathcal{E}}(x) \\ c_{\mathcal{I}}(x) - s \end{bmatrix}.$$

For a sequence of barrier parameters $\mu \downarrow 0$, problem (1.1) is solved through the solution of a sequence of barrier subproblems of the form

$$\min_{z \in \mathbb{R}^{n+q}} \ \varphi(z;\mu), \quad \text{s.t. } c(z) = 0. \tag{2.1}$$

If f, $c_{\mathcal{E}}$, and $c_{\mathcal{I}}$ are continuously differentiable, then first-order Karush-Kuhn-Tucker (KKT) optimality conditions for (2.1) are

$$\begin{bmatrix} \gamma(z;\mu) + A(z)^T \lambda \\ c(z) \end{bmatrix} = 0 \tag{2.2}$$

along with $s \geq 0$, where $\lambda \in \mathbb{R}^{p+q}$ is a vector of Lagrange multipliers, $e \in \mathbb{R}^q$ is a vector of ones,

$$\gamma(z;\mu) := \begin{bmatrix} \nabla f(x) \\ -\mu S^{-1} \Sigma e \end{bmatrix}, \quad \text{and} \quad A(z) := \begin{bmatrix} \nabla c_{\mathcal{E}}(x)^T & 0 \\ \nabla c_{\mathcal{I}}(x)^T & -\Sigma \end{bmatrix}.$$

In situations where (1.1) is infeasible, (2.2) has no solution, so the algorithm is designed to transition automatically from solving (2.1) to solving (1.2), where the latter problem has the KKT conditions

$$A(z)^T c(z) = 0 \tag{2.3}$$

along with $s \geq 0$ and $c_{\mathcal{I}}(x) - s \leq 0$. In fact, the algorithm maintains $s \geq 0$ and $c_{\mathcal{I}}(x) - s \leq 0$ during each iteration by increasing s when necessary. Thus, convergence to a solution of the barrier subproblem (2.1) or an infeasible stationary point of (1.1) is achieved once (2.2) or (2.3), respectively, is satisfied.

At an iterate (z_k, λ_k), the algorithm computes a primal-dual search direction (d_k, δ_k) satisfying appropriate conditions for guaranteeing global convergence; see §2.2. Given such a direction, we compute the scaled search direction $\widetilde{d}_k := (d_k^x, \Sigma_k d_k^s)$ along which a line search is performed. The line search involves two conditions. First, to maintain positivity of the slack variables, a stepsize $\alpha_k^{\max} \in (0, 1]$ satisfying the fraction-to-the-boundary rule

$$s_k + \alpha_k^{\max} \Sigma_k d_k^s \geq (1 - \eta_1) s_k \tag{2.4}$$

is determined for a given constant $\eta_1 \in (0, 1)$. We use $\eta_1 = \max\{0.99, 1 - \mu\}$ in our implementation. The algorithm then backtracks from this value to compute $\alpha_k \in (0, \alpha_k^{\max}]$ yielding sufficient decrease in the penalty function

$$\phi(z;\mu,\pi) := \varphi(z;\mu) + \pi \|c(z)\|, \tag{2.5}$$

where $\pi > 0$ is a penalty parameter. The condition we enforce is

$$\phi(z_k + \alpha_k \widetilde{d}_k; \mu, \pi) \leq \phi(z_k; \mu, \pi) - \eta_2 \alpha_k \Delta m_k(d_k; \mu, \pi) \tag{2.6}$$

where $\eta_2 \in (0, 1)$ is a given constant (we choose $\eta_2 = 10^{-8}$), and where $\Delta m_k(d_k; \mu, \pi)$ is a quantity related to the directional derivative of the penalty function along the computed search direction. We define $\Delta m_k(d_k; \mu, \pi)$ in equation (2.10) in §2.2. In the dual space, we update $\lambda_{k+1} \leftarrow \lambda_k + \beta_k \delta_k$ where β_k is the smallest value in $[\alpha_k, 1]$ satisfying

$$\|\gamma_k + A_k^T \lambda_{k+1}\| \leq \|\gamma_k + A_k^T (\lambda_k + \delta_k)\|. \tag{2.7}$$

That is, we set λ_{k+1} so that the norm of the dual feasibility vector is at least as small as would be obtained for a unit steplength in the dual space.

Our complete interior-point framework is presented as Algorithm 1. In the algorithm, we refer to two search direction computation variants, Algorithms 2 and 4, that are presented in §2.2. These methods include mechanisms for computing (d_k, δ_k) and updating the penalty parameter π. The termination criteria for the original problem (1.1) and the barrier problem (2.1), the choice of the initial point, and the updating rule for the barrier parameter μ are identical with those in [49].

Algorithm 1 Interior-Point Framework

1: (Initialization) Choose line search parameters $\eta_1, \eta_2 \in (0, 1)$, an initial barrier parameter $\mu > 0$, and an initial penalty parameter $\pi > 0$. Initialize (x_0, s_0, λ_0) so that the slack variables satisfy $s_0 > 0$ and $s_0 \geq c_{\mathcal{I}}(x_0)$. Set $k \leftarrow 0$.
2: (Tests for convergence) If convergence criteria for (1.1) are satisfied, then terminate and return x_k as an optimal solution. Else, if convergence criteria for (1.2) are satisfied and x_k is infeasible for (1.1), then terminate and return x_k as an infeasible stationary point.
3: (Barrier parameter update) If convergence criteria for (2.1) are satisfied, then decrease the barrier parameter $\mu > 0$, reset $\pi > 0$, and go to step 2.
4: (Search direction computation) Compute (d_k, δ_k) and update π by Algorithm 2 or Algorithm 4. Set the search direction as $\widetilde{d}_k \leftarrow (d_k^x, \Sigma_k d_k^s)$.
5: (Line search) If $\widetilde{d}_k = 0$, then $\alpha_k \leftarrow 1$. Else, let $\alpha_k^{\max}$ be the largest value in $(0, 1]$ satisfying (2.4) and let l be the smallest value in $\mathbb{N}_0$ such that $\alpha_k \leftarrow 2^{-l} \alpha_k^{\max}$ satisfies (2.6).
6: (Iterate update) Set $z_{k+1} \leftarrow z_k + \alpha_k \widetilde{d}_k$, $s_{k+1} \leftarrow \max\{s_{k+1}, c_{\mathcal{I}}(x_{k+1})\}$, update λ_{k+1} according to (2.7), set $k \leftarrow k + 1$, and go to step 3.

2.2 Search Direction Computation

The main computational component of each iteration of Algorithm 1 is the primal-dual search direction calculation (step 4). We describe two related approaches for computing this direction, presented as Algorithms 2 and 4 in this subsection. The first approach is a simpler calculation, but global convergence for the overall algorithm is only guaranteed with this method if an infinite subsequence of iterations involve (scaled) constraint Jacobians $\{A_k\}$ that have full row rank and singular values bounded away from zero. Otherwise, the second approach must be employed to ensure convergence. The two algorithms have many common features, and we start by discussing the components present in both techniques. (In §3.1 we discuss

our implemented mechanism for having the algorithm dynamically choose between these two approaches during each iteration of Algorithm 1.)

The search direction computation is based on Newton's method applied to the KKT conditions of problem (2.1). Defining the scaled Hessian matrix

$$W(z,\lambda;\mu) := \begin{bmatrix} \nabla^2_{xx} f & 0 \\ 0 & \Sigma\Xi\Sigma \end{bmatrix} + \sum_{i=1}^{p} \lambda_{\mathcal{E}}^{(i)} \begin{bmatrix} \nabla^2_{xx} c_{\mathcal{E}}^{(i)} & 0 \\ 0 & 0 \end{bmatrix} + \sum_{i=1}^{q} \lambda_{\mathcal{I}}^{(i)} \begin{bmatrix} \nabla^2_{xx} c_{\mathcal{I}}^{(i)} & 0 \\ 0 & 0 \end{bmatrix}, \quad (2.8)$$

a Newton iteration for (2.2) is defined by the linear system

$$\begin{bmatrix} W_k & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \delta_k \end{bmatrix} = - \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ c_k \end{bmatrix}. \quad (2.9)$$

In (2.8), the diagonal matrix Ξ is the Hessian (approximation) for the barrier term; see §3.2. We use second derivatives of the objective and constraint functions for the numerical experiments reported in §4 in order to achieve a fast local convergence rate as in Newton's method. However, global convergence can be guaranteed by only requiring that W_k is a uniformly bounded real symmetric matrix.

The central issue that must be confronted when applying Newton's method for large-scale applications is that exact solutions of (2.9) are computationally expensive to obtain. Therefore, our major concern is how an iterative linear system solver can be employed for solving (2.9) in such a way that inexact solutions are allowed, yet global convergence of the algorithm is guaranteed. This issue was the inspiration for all of the algorithms proposed in [13, 14, 19, 20], and the approaches described below are derived from these methods.

Algorithms 2 and 4 each outline a series of termination tests for an iterative solver applied to (2.9) that state conditions under which an inexact solution (d_k, δ_k) can be considered an acceptable direction for step 4 in Algorithm 1. In the following bullets we define the quantities that appear in these tests.

- Let the dual residual vector corresponding to the first block of equations in (2.9) be
$$\rho_k(d_k, \delta_k) := \gamma_k + W_k d_k + A_k^T(\lambda_k + \delta_k).$$
If the linear system (2.9) is solved exactly, then this quantity is zero, but in our tests we only require that it is sufficiently small in norm.
- Let the constraint residual vector corresponding to the second block of equations in (2.9) be
$$r_k(d_k) := c_k + A_k d_k.$$
Again, if (2.9) is solved exactly, then this quantity is zero, but in our tests we only require that it is relatively small in norm. In some cases, this is enforced implicitly via other conditions.
- Let the overall primal-dual relative residual be
$$\Psi_k(d_k, \delta_k) := \left\| \begin{bmatrix} \rho_k(d_k, \delta_k) \\ r_k(d_k) \end{bmatrix} \right\| \Big/ \left\| \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ c_k \end{bmatrix} \right\|.$$
To promote fast convergence, this relative residual should be small [21]. Thus, our implementation aims to compute steps for which this relative residual is below a desired threshold. If the iterative linear system solver is unable to

achieve this accuracy after a given number of iterations, however, then we remove this restriction and are content with potentially less accurate solutions that satisfy our remaining termination criteria. In addition, our test for triggering modifications to the Hessian matrix (to ensure descent) is only considered if $\Psi_k(d_k, \delta_k)$ is small, since otherwise unnecessary Hessian modifications may be made based on inaccurate intermediate solutions in the iterative solver.

For convex problems, the optimization algorithm can focus exclusively on Ψ_k, terminating the calculation of (d_k, δ_k) whenever this value is below a threshold [21]. For nonconvex problems, however, the priority is to find solutions to (2.2) that correspond to minimizers of the optimization problem and not saddle points or maximizers. The methods developed in [13,14,19,20] therefore include additional conditions and procedures that aid the algorithms in converging toward minimizers of (2.1). These additional conditions involve the following quantities.

– Let a local model of the penalty function $\phi(z; \mu, \pi)$ at z_k be

$$m_k(d; \mu, \pi) := \varphi_k + \gamma_k^T d + \pi \|c_k + A_k d\|.$$

The reduction in this model yielded by a given d_k is

$$\begin{aligned} \Delta m_k(d_k; \mu, \pi) &:= m_k(0; \mu, \pi) - m_k(d_k; \mu, \pi) \\ &= -\gamma_k^T d_k + \pi(\|c_k\| - \|c_k + A_k d_k\|). \end{aligned} \tag{2.10}$$

It can be shown (see [20]) that this quantity yields

$$D\phi_k(\widetilde{d}_k; \mu, \pi) \leq -\Delta m_k(d_k; \mu, \pi),$$

where $D\phi_k(\widetilde{d}_k; \mu, \pi)$ is the directional derivative of $\phi(z; \mu, \pi)$ at z_k along $\widetilde{d}_k$. To ensure that $\widetilde{d}_k$ is always a descent direction for the merit function, the termination tests require that this model reduction is sufficiently large, potentially after an increase of π.

We are now prepared to present our first approach, given as Algorithm 2 below. The algorithm contains three termination tests that are similar in spirit and form to those contained in Algorithm 4 later on. The first states that a given (d_k, δ_k) is acceptable if it corresponds to a sufficiently accurate solution to (2.9) and yields a sufficiently large reduction $\Delta m_k(d_k; \mu, \pi)$ for the most recent value of the penalty parameter π. Generally speaking, this condition is the one expected to be satisfied most often in a run of Algorithm 1. The second test corresponds to situations when the first test may not be satisfied by the current value of π, and so it includes conditions under which we allow an increase in this value. Finally, the third condition is only necessary to allow an update of the multipliers if the primal variables are already (nearly) optimal. In [13,14,19,20], these tests are called sufficient merit function approximation reduction termination tests (SMART tests, for short) due to the significant role that $\Delta m_k(d_k; \mu, \pi)$ plays in the theoretical behavior the algorithm. For our numerical experiments, we choose $J = 100$, $\kappa_{\text{des}} = 10^{-3}$, $\kappa = 10^{-2}$, $\epsilon_1 = 0.09$, $\theta = 10^{-12}\mu$ (where μ is the current value of the barrier parameter), $\zeta = 10^{-4}$, $\tau = 0.1$, $\kappa_3 = 10^{-3}$, $\epsilon_3 = 10^{-8}$, and $\kappa_W = 10^{-2}$. In particular, we typically require the relative residual to be only less

than $\kappa = 10^{-2}$ or $\kappa_{\text{des}} = \kappa_3 = 10^{-3}$, but the actual tolerance is dictated by the collection of the entire termination criteria. Note that we choose the parameter θ in the curvature condition in step 6 to depend linearly on μ, because W_k contains the scaled Hessian of the barrier term, $\Sigma_k \Xi_k \Sigma_k$; in a primal barrier method this matrix equals μI. See §3.2 for our choices of ξ and D_k.

The assumption on the iterative linear system solver (see step 7) is that it computes a sequence of search directions such that, in the limit, $\Psi_k \to 0$. Then it can be shown that Algorithm 2 (and Algorithm 4 below) are finite. The method also dynamically modifies W_k during the computation to ensure descent properties of the search direction; see [14,19,20] for motivation and §3.2 for details. In particular, we assume that $W_k \succeq 2\theta I$ after a finite number of modifications.

Algorithm 2 Inexact Newton Iteration with SMART Tests

1: (Initialization) Choose parameters $J \in \mathbb{N}_0$, $\kappa_{\text{des}} \in (0,1)$, $\kappa \in (0,1)$, $\epsilon_1 \in (0,1)$, $\theta > 0$, $\zeta > 0$, $\tau \in (0,1)$, $\kappa_3 \in (0,1)$, $\epsilon_3 > 0$, $\kappa_W > 0$, and $\xi > 0$. Choose a diagonal matrix $D_k \succ 0$. Initialize $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0,0)$.

2: (Residual test) If $j \leq J$ and $\Psi_k > \kappa_{\text{des}}$, then go to step 7.

3: (Termination test 1) If $\Psi_k \leq \kappa$ and the model reduction condition

$$\Delta m_k(d_k; \mu, \pi) \geq \max\{\tfrac{1}{2} d_k^T W_k d_k, \theta \|d_k\|^2\} + \epsilon_1 \pi \max\{\|c_k\|, \|r(d_k)\| - \|c_k\|\} \tag{2.11}$$

holds, then terminate by returning (d_k, δ_k) and the current π.

4: (Termination test 2) If the residual conditions

$$\begin{aligned} \|\rho_k(d_k, \delta_k)\| &\leq \kappa \|c_k\| \\ \|r_k(d_k)\| &\leq \kappa \|c_k\| \end{aligned}$$

are satisfied and the curvature condition $\frac{1}{2} d_k^T W_k d_k \geq \theta \|d_k\|^2$ holds, then terminate by returning (d_k, δ_k) and $\pi \leftarrow \max\{\pi, \pi^t + \zeta\}$ where

$$\pi^t \leftarrow \frac{\gamma_k^T d_k + \frac{1}{2} d_k^T W_k d_k}{(1-\tau)(\|c_k\| - \|r_k(d_k)\|)}.$$

5: (Termination test 3) If the dual displacement δ_k yields

$$\|\rho_k(0, \delta_k)\| \leq \kappa_3 \|\gamma_k + A_k^T \lambda_k\|$$

and the primal and dual feasibility measures satisfy

$$\|c_k\| \leq \epsilon_3 \|\gamma_k + A_k^T \lambda_k\|,$$

then terminate by returning $(0, \delta_k)$ (i.e., reset $d_k \leftarrow 0$) and the current π.

6: (Hessian modification) If $\Psi_k \leq \kappa_W$ and $\frac{1}{2} d_k^T W_k d_k < \theta \|d_k\|^2$, then modify $W_k \leftarrow W_k + \xi D_k$, reset $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0,0)$, and go to step 2.

7: (Search direction update) Perform one iteration of an iterative solver on (2.9) to compute an improved (approximate) solution (d_k, δ_k). Increment $j \leftarrow j + 1$ and go to step 2.

Our second approach, Algorithm 4, can be viewed as a replacement for Algorithm 2 in situations when the constraint Jacobian may be ill-conditioned or rank-deficient. In such cases, it is necessary to regularize the search direction computation since otherwise the calculation may not be well-defined, or at best may lead to long, unproductive search directions. Moreover, in order to guarantee global

convergence, the algorithm must avoid scenarios such as that described in a counterexample to the convergence of certain interior-point algorithms [47].

Algorithm 4 performs these tasks by decomposing the primal search direction as $d_k := v_k + u_k$, where the *normal component* v_k represents a direction toward linearized feasibility and the *tangential component* u_k represents a direction toward optimality. The normal component v_k is defined as an approximate solution to the subproblem

$$\begin{aligned} \min_{v\in\mathbb{R}^n} \ & \tfrac{1}{2}\|r_k(v)\|^2 \\ \text{s.t. } & \|v\| \le \omega\|A_k^T c_k\| \end{aligned} \tag{2.12}$$

for some $\omega > 0$. The trust region constraint regularizes the computation of the normal step and controls the size of this component even when A_k loses rank. We initially choose $\omega = 100$ in our implementation, but have found it practically beneficial to potentially increase this value at the end of iteration k according to the following rule (see [19]):

$$\omega \leftarrow \begin{cases} \min\{10\omega, 10^{20}\} & \text{if } \|v_k\| = \omega\|A_k^T c_k\| \text{ and } \alpha_k = 1, \\ \omega & \text{otherwise.} \end{cases}$$

This rule increases ω if the algorithm accepts a unit steplength for a step whose normal component hits the trust region boundary, suggesting that the trust region constraint may be impeding fast convergence.

As with the linear system (2.9), an exact solution of (2.12) is computationally expensive to obtain. However, global convergence is guaranteed as long as v_k is feasible for problem (2.12) and satisfies the Cauchy decrease condition

$$\|c_k\| - \|r(v_k)\| \ge \epsilon_v(\|c_k\| - \|r(\bar{\alpha}_k\bar{v}_k)\|) \tag{2.13}$$

for some constant $\epsilon_v \in (0,1)$ (we choose $\epsilon_v = 0.1$); see [19,20]. The vector $\bar{v}_k := -A_k^T c_k$ is the steepest descent direction for the objective of problem (2.12) at $v = 0$ and the steplength $\bar{\alpha}_k$ is the solution to the one-dimensional optimization problem

$$\min_{\bar{\alpha}\in\mathbb{R}} \ \tfrac{1}{2}\|c_k + \bar{\alpha}A_k\bar{v}_k\|^2, \quad \text{s.t. } \bar{\alpha} \le \omega. \tag{2.14}$$

A number of techniques have been developed and analyzed for the inexact solution of large-scale instances of problem (2.12) with solutions satisfying a Cauchy decrease condition; e.g., see the conjugate-gradient method described in [44]. In our software, we implemented Algorithm 3 below, which is a type of inexact dogleg approach [35,36] and was also proposed in [20]. We begin by computing the Cauchy point $v_k^C := \bar{\alpha}_k\bar{v}_k$, and then (approximately) solve the augmented system (e.g., see [16])

$$\begin{bmatrix} I & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} v_k^N \\ \delta_k^N \end{bmatrix} = - \begin{bmatrix} 0 \\ c_k \end{bmatrix}, \tag{2.15}$$

which for an inexact solution yields the residual vector

$$\Psi_k^N(v_k^N, \delta_k^N) := \begin{bmatrix} \rho_k^N(v_k^N, \delta_k^N) \\ r_k(v_k^N) \end{bmatrix} := \begin{bmatrix} v_k^N + A_k^T\delta_k^N \\ A_k v_k^N + c_k \end{bmatrix}.$$

Note that an exact solution of (2.15) yields the least-norm solution of (2.12) for $\omega = \infty$. The inexact dogleg step is then defined as a point along the line segment

between the Cauchy point and v_k^N that is feasible for problem (2.12) and satisfies the Cauchy decrease condition (2.13). We tailor this approach into one that has worked well in our tests. In particular, we consider the fraction-to-the-boundary rule (2.4) when choosing between the Cauchy point and the inexact dogleg step. The constants defined in the algorithm are set as $\kappa_v = 10^{-3}$ and $\bar{\epsilon}_v = 10^{-10}$ in our implementation, and the iteration limit for the linear solver in step 3 is 200.

Algorithm 3 Normal Component Iteration

1: (Initialization) Choose parameters $\epsilon_v \in (0,1)$, $\kappa_v \in (0,1)$, $\bar{\epsilon}_v > 0$ and $\omega > 0$.
2: (Cauchy point computation) Set $v_k^C = \bar{\alpha}_k \bar{v}_k$ where $\bar{v}_k = -A_k^T c_k$ and $\bar{\alpha}_k$ solves (2.14).
3: (Inexact augmented system solve) Apply an iterative linear system solver to (2.15) to compute (v_k^N, δ_k^N) satisfying $\|\Psi_k^N(v_k^N, \delta_k^N)\| \leq \max\{\kappa_v \|c_k\|, \bar{\epsilon}_v\}$ and $\|r_k(v_k^N)\| \leq \|r_k(v_k^C)\|$ or as the last vector computed before an iteration limit is reached.
4: (Inexact dogleg computation) Set $v_k^D = (1-\alpha)v_k^C + \alpha v_k^N$ where $\alpha \in [0,1]$ is the largest value such that v_k^D is feasible for (2.12). Set α_k^C and α_k^D as the largest values in $[0,1]$ satisfying (2.4) along v_k^C and v_k^D, respectively. If

$$\|c_k\| - \|r_k(\alpha_k^D v_k^D)\| \geq \epsilon_v(\|c_k\| - \|r_k(\alpha_k^C v_k^C)\|),$$

then set $v_k \leftarrow v_k^D$; else, set $v_k \leftarrow v_k^C$.

We are now prepared to present Algorithm 4. Due to the fact that the normal component v_k is computed separately from the tangential component u_k, we now apply an iterative linear system solver to the reformulated system

$$\begin{bmatrix} W_k & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} d_k \\ \delta_k \end{bmatrix} = - \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ -A_k v_k \end{bmatrix} \tag{2.16}$$

where the second block of equations stipulates $A_k d_k = A_k v_k$; i.e., progress toward linearized feasibility is aimed to be similar to that obtained by v_k. The relative residual Ψ_k is redefined accordingly as

$$\Psi_k(d_k, \delta_k) := \left\| \begin{bmatrix} \rho_k(d_k, \delta_k) \\ -A_k v_k + A_k d_k \end{bmatrix} \right\| \Big/ \left\| \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ -A_k v_k \end{bmatrix} \right\|.$$

It is important to note that the second block of equations in (2.16) is consistent, and so the entire system is consistent for suitable W_k. We choose the input parameters for Algorithm 4 to be the same as those used in Algorithm 2. The values for the new constants are chosen to be $\epsilon_2 = 0.9$ and $\psi = 0.1$. Conditions on the dual residual are complicated by the fact that we require information from the previous iteration (see [20]).

3 Algorithm Details

In this section we discuss further algorithmic details and enhancements to the methods described in §2. In particular, we describe our technique for deciding between Algorithms 2 and 4 for each iteration during a run of Algorithm 1, expand on our method for modifying W_k during these two algorithms, discuss the incorporation of a flexible penalty function, and summarize the details of our preconditioning strategy.

Algorithm 4 Regularized Inexact Newton Iteration with SMART Tests

1: (Initialization) Choose parameters $J \in \mathbb{N}_0$, $\kappa_{\text{des}} \in (0,1)$, $\psi > 0$, $\theta > 0$, $\kappa \in (0,1)$, $\epsilon_1 \in (0,1)$, $\epsilon_2 \in (0,1)$, $\zeta > 0$, $\tau \in (0,1)$, $\kappa_3 \in (0,1)$, $\epsilon_3 \in (0,1)$, $\kappa_W > 0$, and $\xi > 0$. Choose a diagonal matrix $D_k \succ 0$. Compute v_k by Algorithm 3. Initialize $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0,0)$.
2: (Residual test) If $j \leq J$ and $\Psi_k > \kappa_{\text{des}}$, then go to step 10.
3: (Direction decomposition) Set $u_k \leftarrow d_k - v_k$.
4: (Tangential component test) If

$$\|u_k\| \leq \psi \|v_k\| \tag{2.17}$$

or if the inequalities

$$\tfrac{1}{2} u_k^T W_k u_k \geq \theta \|u_k\|^2 \tag{2.18a}$$

$$(\gamma_k + W_k v_k)^T u_k + \tfrac{1}{2} u_k^T W_k u_k \leq 0 \tag{2.18b}$$

are satisfied, then continue to step 5; otherwise, go to step 8.
5: (Dual residual test) If the dual residual condition

$$\|\rho_k(d_k, \delta_k)\| \leq \kappa \min\left\{ \left\| \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ A_k v_k \end{bmatrix} \right\|, \left\| \begin{bmatrix} \gamma_{k-1} + A_{k-1}^T \lambda_k \\ A_{k-1} v_{k-1} \end{bmatrix} \right\| \right\} \tag{2.19}$$

is satisfied, then continue to step 6; otherwise, go to step 8.
6: (Termination test 1) If the model reduction condition

$$\Delta m_k(d_k; \mu, \pi) \geq \max\{\tfrac{1}{2} u_k^T W_k u_k, \theta \|u_k\|^2\} + \epsilon_1 \pi (\|c_k\| - \|r_k(v_k)\|) \tag{2.20}$$

is satisfied, then terminate by returning (d_k, δ_k) and the current π.
7: (Termination test 2) If the linearized constraint condition

$$\|c_k\| - \|r_k(d_k)\| \geq \epsilon_2 (\|c_k\| - \|r_k(v_k)\|) > 0$$

is satisfied, then terminate by returning (d_k, δ_k) and $\pi \leftarrow \max\{\pi, \pi^t + \zeta\}$ where

$$\pi^t \leftarrow \frac{\gamma_k^T d_k + \frac{1}{2} u_k^T W_k u_k}{(1-\tau)(\|c_k\| - \|r_k(d_k)\|)}.$$

8: (Termination test 3) If the dual displacement δ_k yields

$$\|\rho_k(0, \delta_k)\| \leq \kappa_3 \min\left\{ \left\| \begin{bmatrix} \gamma_k + A_k^T \lambda_k \\ A_k v_k \end{bmatrix} \right\|, \left\| \begin{bmatrix} \gamma_{k-1} + A_{k-1}^T \lambda_k \\ A_{k-1} v_{k-1} \end{bmatrix} \right\| \right\}$$

and the stationarity and dual feasibility measures satisfy

$$\|A_k^T c_k\| \leq \epsilon_3 \|\gamma_k + A_k^T \lambda_k\|,$$

then terminate by returning $(0, \delta_k)$ (i.e., reset $d_k \leftarrow 0$) and the current π.
9: (Hessian modification) If $\Psi_k \leq \kappa_W$, but both (2.17) and (2.18a) do not hold, then modify $W_k \leftarrow W_k + \xi D_k$, reset $j \leftarrow 1$ and $(d_k, \delta_k) \leftarrow (0,0)$, and go to step 2.
10: (Search direction update) Perform one iteration of an iterative solver on (2.16) to compute an improved (approximate) solution (d_k, δ_k). Increment $j \leftarrow j + 1$ and go to step 2.

3.1 Switching between search direction calculations

Algorithm 1 paired with Algorithm 4 constitutes an approach that is theoretically globally convergent to first-order stationary points under common assumptions [20]. However, as Algorithm 2 will produce viable search directions in most practical situations and, in contrast to Algorithm 4, it only requires the inexact solution

of a single linear system, it is generally advantageous to pair Algorithm 1 with Algorithm 2 rather than with Algorithm 4. Thus, our implementation computes search directions with Algorithm 2 and only switches to Algorithm 4 when there is evidence that Algorithm 2 may be unable to produce a productive search direction.

Our trigger for switching between the two search direction algorithms is based on the steplength obtained as a result of the line search. If during a given iteration of Algorithm 1, Algorithm 2 has been employed for computing the search direction and the line search produces a steplength below a given threshold $\bar{\alpha}_1$, then this may be an indication that A_k is losing rank, causing the steps to become too large. (Of course, the short steplength may simply be due to the nonlinearity of the problem functions themselves, but even in that case the algorithm may benefit by employing Algorithm 4.) In such cases, we decide to employ Algorithm 4 in the following iteration of Algorithm 1 and continue to employ it until an iteration yields a steplength above $\bar{\alpha}_1$. The motivation for this choice is that Algorithm 1 paired with Algorithm 2 is guaranteed to converge for an equality-constrained problem (e.g., a given barrier subproblem) under common assumptions, including that the constraint Jacobians have full row rank and their smallest singular values are bounded away from zero. Specifically, for the analysis of Algorithm 2 in [14], the latter assumption is used to show that the steplength α_k is bounded away from zero. Thus, we use a small steplength α_k as an indicator to switch to Algorithm 4. In our implementation, we choose the threshold value to be $\bar{\alpha}_1 = 10^{-3}$.

We have also included a fall-back mechanism in our implementation that causes the algorithm to switch from Algorithm 2 to Algorithm 4 during an iteration of Algorithm 1 if the iterative linear system solver fails to produce an acceptable search direction even after tightening tolerances for the preconditioner (see §4).

3.2 Hessian modification strategy

In the definition of the Hessian matrix W_k in (2.8), the choice $\Xi_k = \mu S_k^{-2}$ corresponds to the so-called *primal* interior-point iteration. This was considered for the global convergence analysis in [20]. However, our implementation follows the more efficient *primal-dual* interior-point strategy $\Xi_k = S_k^{-1} Y_k$, where $Y_k = \text{diag}(y_k)$ with dual variables y_k corresponding to the slack bounds (i.e. $s \geq 0$). It is easy to see that the analysis in [20] still holds as long as the condition

$$\overline{\nu}\mu S_k^{-2} \succeq \Xi_k \succeq \underline{\nu}\mu S_k^{-2} \tag{3.1}$$

is satisfied for some constants $\overline{\nu} \geq 1 \geq \underline{\nu} > 0$; we choose $\overline{\nu} = 10^{10}$ and $\underline{\nu} = 10^{-10}$ in our experiments. This is achieved by adjusting y_k if necessary after each iteration to ensure (3.1), as described in [49].

Our strategy for modifying the Hessian in Algorithms 2 and 4 is analogous to the one described in [49], where a multiple of the identity is added to the *unscaled* Hessian matrix. This corresponds to using

$$D_k = \begin{bmatrix} I & 0 \\ 0 & \Sigma_k^2 \end{bmatrix}.$$

Furthermore, we choose ξ according to the strategy for choosing δ_w in Algorithm IC in [49]. However, in contrast to Algorithm IC, our trigger for a modification is not

the inertia of the primal-dual system. Rather, we trigger a modification based on the conditions described in step 6 of Algorithm 2 and step 9 of Algorithm 4. We have also found it beneficial in our numerical experiments to trigger a modification at the start of the search direction computation if in the previous iteration the line search reduced α_k due to the sufficient decrease condition (2.6). This leads to somewhat shorter search directions and makes the acceptance of larger steplengths α_k more likely, often leading to a reduction in iteration count.

3.3 Flexible penalty function

An important algorithmic feature of our code is the use of a flexible penalty function [18]. This mechanism is designed to avoid a pitfall of penalty functions, namely the potential for the algorithm to set an unnecessarily large value of the penalty parameter and thus restricting the iterates to remain close to the feasible region. This can lead to small steplengths and slow convergence.

The effect of the flexible penalty function on our line search is that, instead of requiring α_k to satisfy the sufficient decrease condition (2.6) for a *fixed* π, we only require that α_k satisfies a sufficient decrease condition for *some* π in an interval $[\pi^l, \pi^u]$. In particular, given $\pi^l \leq \pi^m \leq \pi^u$, $\alpha_k \in (0, \alpha_k^{\max}]$ is acceptable as long as

$$\begin{aligned} \phi(z_k + \alpha_k \widetilde{d}_k; \mu, \pi) \leq\ & \phi(z_k; \mu, \pi) - \eta_2 \alpha_k \Delta m_k(d_k; \mu, \pi^m) \\ \text{for some } \pi \in\ & [\pi^l, \pi^u]. \end{aligned} \tag{3.2}$$

Generally speaking, π^l and π^u can be set to relatively small and large values, respectively, and π^m need only be set large enough so that the model reduction $\Delta m_k(d_k; \mu, \pi^m)$ is sufficiently positive. However, all of these values need to be updated carefully in order to ensure convergence.

We have adapted the strategy proposed in [18] for updating π^l and π^u and for setting π^m during each iteration. In fact, π^u is intended to play the role of the original penalty parameter value as presented in Algorithms 2 and 4 (except in the model reduction conditions (2.11) and (2.20)), so the update for this value is the same as that for π in Termination test 2 of each of these algorithms. The only difference is that, in the context of a flexible penalty function, π^u can be initialized to a larger value than one would normally initialize the penalty parameter in other approaches; in our experiments, we initialize $\pi^l \leftarrow 10^{-6}$ and $\pi^u \leftarrow 1$. The value π^l, as in [18], is designed to reflect the changes in the *nonlinear* (i.e., not the model) functions as the overall algorithm proceeds. We only update this quantity after the line search whenever (3.2) failed to hold for $\pi = \pi^l$. In such cases, we update

$$\pi^l \leftarrow \min\{\pi^u, \pi^l + \max\{10^{-4}(\chi - \pi^l), 10^{-4}\}\},$$

where

$$\chi := \frac{\phi(z_k + \alpha_k \widetilde{d}_k; \mu) - \phi(z_k; \mu)}{\|c(z_k)\| - \|c(z_k + \alpha_k \widetilde{d}_k)\|}.$$

After a repeated number of updates of this form, the algorithm could potentially set $\pi^l = \pi^u$, but this is rare in practice, meaning that in general the flexible penalty function provides a much wider variety of steps to be acceptable to the line search than a classical penalty function.

There are two more issues to consider regarding the use of a flexible penalty function. One concerns the value of π that appears in the model reduction conditions (2.11) and (2.20). For this value, we choose π^l so that either condition will be satisfied more easily. The second issue is the choice of π^m, for which we consider two cases. If in the current iteration Termination test 2 was satisfied, but Termination test 1 was not, then we follow [18] and set π^m to be the maximum of π^l and π^t, where π^t is computed during Termination test 2. This choice guarantees that the model reduction $\Delta m_k(d_k; \mu, \pi^m)$ is sufficiently positive. Otherwise, if Termination Test 1 was satisfied, then we set π^m to be π^l since, based on our decision to use π^l as the penalty parameter value in (2.11) and (2.20), this choice also guarantees that the model reduction $\Delta m_k(d_k; \mu, \pi^m)$ is positive. Overall, in either case, we guarantee that (3.2) is a sufficient decrease condition for ϕ.

3.4 Preconditioning

When an iterative solver is used to solve the linear systems (2.9), (2.15), and (2.16), an effective preconditioner is essential to keep the number of iterations low. For our numerical experiments, we use the general-purpose preconditioning approach described in this section. We remark, however, that for best performance, preconditioners should be tailored to particular problems.

Our approach is based on an algebraic multilevel preconditioner recently designed for symmetric highly indefinite systems. It uses symmetric maximum weight matchings to improve the block diagonal dominance of the system, followed by an inverse-based pivoting strategy to compute an inexact factorization. In order to bound the norm of the inverse, the factorization of some rows and columns might be postponed to the end. This leaves a Schur complement to which the procedure is applied recursively within a multilevel preconditioning framework.

Details of the preconditioning method can be found in [39,42]. Here, we only give a short summary of the procedure. We denote $\bar{A} \in \mathbb{R}^{m \times m}$ as the matrix in the linear system of interest, whether it be (2.9), (2.15), or (2.16).

Symmetric weighted matching can be viewed as a preprocessing step that computes a positive diagonal matrix $D \in \mathbb{R}^{m \times m}$ and a permutation matrix $P \in \mathbb{R}^{m \times m}$ to obtain a new matrix

$$\hat{A} = P^T D \bar{A} D P. \tag{3.3}$$

All entries $\hat{a}^{(i,j)}$ of $\hat{A}$ are at most one in size. Moreover, the diagonal blocks are either 1×1 scalars $\hat{a}^{(i,i)}$ with $|\hat{a}^{(i,i)}| = 1$ (in exceptional cases one has $\hat{a}^{(i,i)} = 0$) or symmetric 2×2 blocks

$$\begin{pmatrix} \hat{a}^{(i,i)} & \hat{a}^{(i+1,i)} \\ \hat{a}^{(i+1,i)} & \hat{a}^{(i+1,i+1)} \end{pmatrix} \text{ such that } |a^{(i,i)}|, |\hat{a}^{(i+1,i+1)}| \leq 1 \text{ and } |\hat{a}^{(i+1,i)}| = 1.$$

In addition, a further permutation based on nested dissection [32] (w.l.o.g. included in P) is computed to reduce the fill-in in a sparse LDL^T factorization. Numerical experiments in [39,41,43] indicate that this preprocessing step, in the context of computing a factorization of $\bar{A}$, makes dynamic pivoting strategies unnecessary.

Once the system is reordered and rescaled according to (3.3), the preconditioner is computed by an incomplete factorization $LDL^T = \hat{A} + E$ of $\hat{A}$. Suppose that at

step j of the factorization algorithm we have

$$\hat{A} = \begin{pmatrix} B & F^T \\ F & C \end{pmatrix} = \begin{pmatrix} L_B & 0 \\ L_F & I \end{pmatrix} \begin{pmatrix} D_B & 0 \\ 0 & S_C \end{pmatrix} \begin{pmatrix} L_B^T & L_F^T \\ 0 & I \end{pmatrix},$$

where $L_B \in \mathbb{R}^{j\times j}$ is unit lower triangular, $D_B \in \mathbb{R}^{j\times j}$ is block diagonal with diagonal blocks of size 1×1 and 2×2, and $S_C = C - L_F D_B L_F^T$ denotes the approximate Schur complement. Following [11,40], one can easily estimate

$$\left\| \begin{pmatrix} L_B & 0 \\ L_F & I \end{pmatrix}^{-1} \right\| \leq \kappa_L \tag{3.4}$$

in every step for a prescribed bound $\kappa_L > 1$ based on a sparse adaption of the method presented in [17]. If at step j the approximate factorization fails to satisfy (3.4), then row and column j are permuted to the end. Otherwise we proceed with the approximate factorization where small entries in L are dropped according to a relative drop tolerance $\epsilon_L \in (0, 1]$. When the approximate LDL^T decomposition has finally passed all rows and columns of $\hat{A}$, we are faced with a system of the form

$$Q^T \hat{A} Q = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I \end{pmatrix} \begin{pmatrix} D_{11} & 0 \\ 0 & S_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & I \end{pmatrix}.$$

The Schur complement S_{22}, corresponding to all postponed updates, is then computed explicitly, and the strategies for reordering, scaling and factoring are recursively applied to S_{22}, leading to a multi-level factorization. To improve sparsity, small elements of the Schur complement S_{22} are dropped before the recursion according to a relative drop tolerance $\epsilon_S \in (0, 1]$. Once the Schur complement has a sufficiently small size (less than 5000 rows and columns in our implementation) or a maximum number of levels is reached, the Schur complement is factored exactly.

For the iterative solution of linear systems we use the symmetric quasi-minimum residual (SQMR) method [24] which has been found to work well with this preconditioner. Here, we allow a depth up to 30 in the multi-level approach, the constant bounding the norm of the inverse of the factor in (3.4) is chosen to be $\kappa_L = 2$, and the drop tolerances for the factor and the Schur complement are set to be $\epsilon_L = 10^{-2}$ and $\epsilon_S = 10^{-3}$, respectively. The SQMR method is allowed a maximum number of 1500 iterations. If this number is exceeded, then the preconditioner is recomputed with tightened drop tolerances (both divided by 3) and the iteration counter is reset. If necessary, the tolerances are tightened repeatedly. If an acceptable solution for Algorithm 2 has not been computed after 4 such attempts, then the method reverts to Algorithm 4. If an acceptable solution for Algorithm 4 has not been computed after 4 such attempts, then the last computed inexact solution is used (without guarantees for a successful line search). In either of these latter two cases, before a new linear system is solved, the drop tolerances are multiplied by 3, though they are never set higher than the default values given above.

4 Numerical Experiments

The algorithm described in the previous sections was implemented in the `IPOPT` open-source optimization package[1]; for our experiments we use revision 1954 of

[1] http://www.coin-or.org/Ipopt/

the `branches/parallel` development branch. The linear systems are solved using the iterative linear system solvers and preconditioners implemented in the `PARDISO` software package[2] version 4.1.1. The finite-element discretization of the PDEs in §4.2-4.3 was implemented using the open-source `libmesh` library [33][3], revision 3881 in its `trunk` branch, together with the PETSc library [3][4] version 3.1-p3. The three-dimensional meshes for the example in §4.3 are generated with the `tetgen` software[5].

In `IPOPT`, we use the default parameter settings with a termination tolerance 10^{-6}, together with the parameter choices given in the previous sections. The iterative linear solver in `PARDISO` uses the SQMR algorithm [24] with the preconditioner described in §3.4.

The numerical experiments in the following sections illustrate the performance of our implementation on a large nonlinear optimization test set and on two PDE-constrained problems. Overall, we show that the method is robust and provides improved computation times compared to the default `IPOPT` algorithm as problem sizes grow large. The results were obtained on 8-core Intel Xeon machines with 2.33GHz clock speed and 32GB RAM, running Ubuntu Linux with GNU 4.4.1 compilers. To avoid tainted CPU times caused by memory bus contention, we ran only one serial process at a time.

4.1 Standard Nonlinear Programming Test Sets

To assess the robustness of the algorithm we compare its performance with the default method implemented in `IPOPT` on problems from the CUTEr test set [25, 26] for which AMPL models [23] are available.[6] We include all feasible problems that have at least one degree of freedom, are not unbounded, and which do not have inequality constraints with both lower and upper bounds in the formulation (the latter is a purely superficial limitation of our current implementation).

Since these problems are not very large, we changed the setting for the `PARDISO` preconditioner so that the multi-level strategy continues until the Schur complement matrix reaches the size 10 (instead of the default 5000). This has the effect that we will always obtain a multilevel iterative preconditioner for matrices that have more than 10 equations. Allowing a CPU-time limit of 30 minutes and a limit of 3000 `IPOPT` iterations, the default algorithm in `IPOPT` [49] using a direct factorization with `PARDISO` and a filter line-search procedure is able to find a point satisfying the termination criteria in 592 out of a total of 617 optimization problems, giving a success rate of 96%. Note that some of the problems do not satisfy the regularity assumptions made for the global convergence analysis in [48]. Failures are also due to exceeding the iteration limit (12 cases), and to numerical issues caused by ill-conditioning. The CPU time limit was not reached for any problem by the default algorithm.

Algorithm 1 terminated successfully for 549 problems (89% success rate), exceeding the iteration limit in 12 and the CPU time limit in 20 cases. In the majority

[2] http://www.pardiso-project.org/
[3] http://libmesh.sourceforge.net/
[4] http://www.mcs.anl.gov/petsc/
[5] http://tetgen.berlios.de/
[6] http://orfe.princeton.edu/~rvdb/ampl/nlmodels/cute/

of the remaining cases, the algorithm broke down because no suitable preconditioner could be computed and the iterative linear system solver did not converge. As the scope of this paper does not include convergence issues of the linear system solver, we did not explore this issue in further detail. We note that for 142 problems, the algorithm switched to Algorithm 4 at some point.

We also compare this performance with that of the original algorithm in [20], which always uses Algorithm 4 (i.e., it decomposes the step computation) in every `IPOPT` iteration. That method was successful for only 518 problems, yielding a success rate of 84%. The iteration limit was exceeded in 18 cases, and the CPU time limit was hit for 33 problems. This demonstrates that an increase in robustness was obtained for our implementation with the addition of the switching strategy described in §3.1. Specifically, the switching strategy increased the percentage of successful solves from 84% to 89%.

We also note that our algorithm is able to solve the counterexample from [47] in 20 iterations, reverting to Algorithm 4 twice (for one iteration each time), following the adaptive step computation strategy described in §3.1. By contrast, the algorithm fails if steps are always computed using Algorithm 2 because the stepsizes α_k converge to zero, as expected from the analysis in [47].

4.2 Optimal boundary control

Our first PDE-constrained optimization problem is an optimal control problem motivated by the "Heating with radiation boundary conditions" example in Section 1.3.1 of [45]:

$$\min_{u,T} \int_\Gamma u \, da$$

subject to

$$\begin{aligned} -\Delta T &= 0 && \text{in } \Omega && \text{(4.1a)}\\ \frac{\partial T}{\partial n} &= \alpha(u - T^4) && \text{on } \Gamma && \text{(4.1b)}\\ T &\geq T_j^{\min} && \text{in } \Omega_j \text{ for } j = 1, \ldots, N_S && \text{(4.1c)}\\ u &\geq 0 && \text{on } \Gamma. && \text{(4.1d)} \end{aligned}$$

Here, T denotes temperature in a domain $\Omega \subseteq \mathbb{R}^3$, and the term $\frac{\partial T}{\partial n}$ denotes the outward-pointing normal derivative of the temperature on the boundary Γ of Ω. The boundary condition (4.1b) expresses the radiation heat loss according to the Stefan-Boltzmann law with a Stefan's constant $\alpha > 0$, where the control u dictates heat that can be resupplied on Γ. The goal is to minimize the amount of heat supplied while attaining a temperature of at least $T_i^{\min}$ within N_S subregions $\Omega_j \subseteq \Omega$. Following the common finite element approach, we multiply (4.1a) with a test function $v \in H^1(\Omega)$ and apply Green's formula together with (4.1b). The weak formulation of the PDE is then to find $T \in H^1(\Omega)$ such that

$$0 = -\int_\Omega \Delta T v \, dx = \int_\Omega \nabla T \cdot \nabla v \, dx - \alpha \int_\Gamma (T^4 - u)\, v \, da \quad \forall v \in H^1(\Omega). \tag{4.2}$$

We generate a regular mesh of tetrahedrons, each with volume $h^3/24$ for a discretization parameter $h > 0$, and use the standard linear finite element basis

h	#var	#bds	#eq	#ineq
0.05	47263	5724	42461	0
0.04	89453	9183	81951	0
0.03	199389	16498	186319	0
0.02	670153	42313	640151	0

Table 4.1 Problem sizes for instances of the boundary control example.

functions $\{\varphi_i\}_{i=1,\ldots,n_h}$. Projecting (4.2) onto the generated finite dimensional subspace V^h by approximating T with $T^h = \sum_i T^{(i)}\varphi_i$ and u by $u^h = \sum_i u^{(i)}\varphi_i$ (the latter requires discretized values $u^{(i)}$ only corresponding to the boundary Γ), we solve the finite-dimensional problem

$$\min_{u^{(i)},T^{(i)}} \sum_i u^{(i)} \int_\Gamma \varphi_i \, da$$

subject to

$$\begin{aligned} 0 = & \int_\Omega \sum_i T^{(i)} \nabla\varphi_i \cdot \nabla\varphi_j \, dx \\ & - \alpha \int_\Gamma \left((\sum_i T^{(i)}\varphi_i)^4 - \sum_i u^{(i)}\varphi_i \right) \varphi_j \, da \quad \text{for } j = 1,\ldots,n_h \qquad (4.3a) \\ T^{(i)} \geq & \; T_j^{\min} \quad \text{for } j \in \{1,\ldots,N_S\} \text{ and } i \in \{\hat{i} \mid \exists x \in \Omega_j : \varphi_{\hat{i}}(x) = 1\} \qquad (4.3b) \\ u^{(i)} \geq & \; 0. \qquad (4.3c) \end{aligned}$$

We choose $\alpha = 1$ and $\Omega = (0,1)^3$ and define two regions to be heated, $\Omega_1 = [0.1, 0.2] \times [0.05, 0.3] \times [0, 0.1]$ and $\Omega_2 = [0.8, 1] \times [0.75, 1] \times [0.7, 1]$, with associated threshold temperatures of $T_1^{\min} = 2.5$ and $T_2^{\min} = 2$. In (4.3b), we used the fact that a nodal finite element basis was chosen, so that $\max_{x\in\Omega} \varphi_i(x) = 1$, and for all $x \in \Omega$ we have $\sum_i \varphi_i(x) = 1$. Since $\nabla\varphi_i \cdot \nabla\varphi_j = \mathcal{O}(1/h^2)$ and $\int_E dx = \mathcal{O}(h^3)$ for a tetrahedron E, we multiply (4.3a) by $10^{-2}/h$ in our implementation, to ensure that the gradients of these constraints do not vanish as $h \to 0$. Similarly, the objective function was scaled internally by the factor $10^{-2}/(h^2)$.

We executed our implementation of the optimization algorithm for four choices of the discretization level. As initial point, we chose $T = T_{\text{init}}$ with $T_{\text{init}} = 1.1(T_1^{\min} + T_2^{\min})$, and $u = (T_{\text{init}})^4$. Table 4.1 shows the discretization parameter (h), number of optimization variables (#var), number of simple bound constraints (#bds), number of equality constraints (#eq), and number of inequality constraints (#ineq) for various instances of this example. Tables 4.2 and 4.3 provide performance measures in the form of number of iterations (it), final objective value $f(x_*)$, CPU seconds (CPUs), and CPU seconds per iteration (CPUs/it) for the default `IPOPT` algorithm and for the new algorithm using inexact steps, respectively. The last column in Table 4.3 shows the overall CPU time speedup of the inexact algorithm compared to the default method. Figure 4.1 shows the optimal solution for the finest discretization $h = 0.02$.

We clearly see a significant gain in computation speed that becomes more pronounced as the problem size increases. For the largest problem with a discretization parameter $h = 0.02$, the speedup is a factor of 7.85.

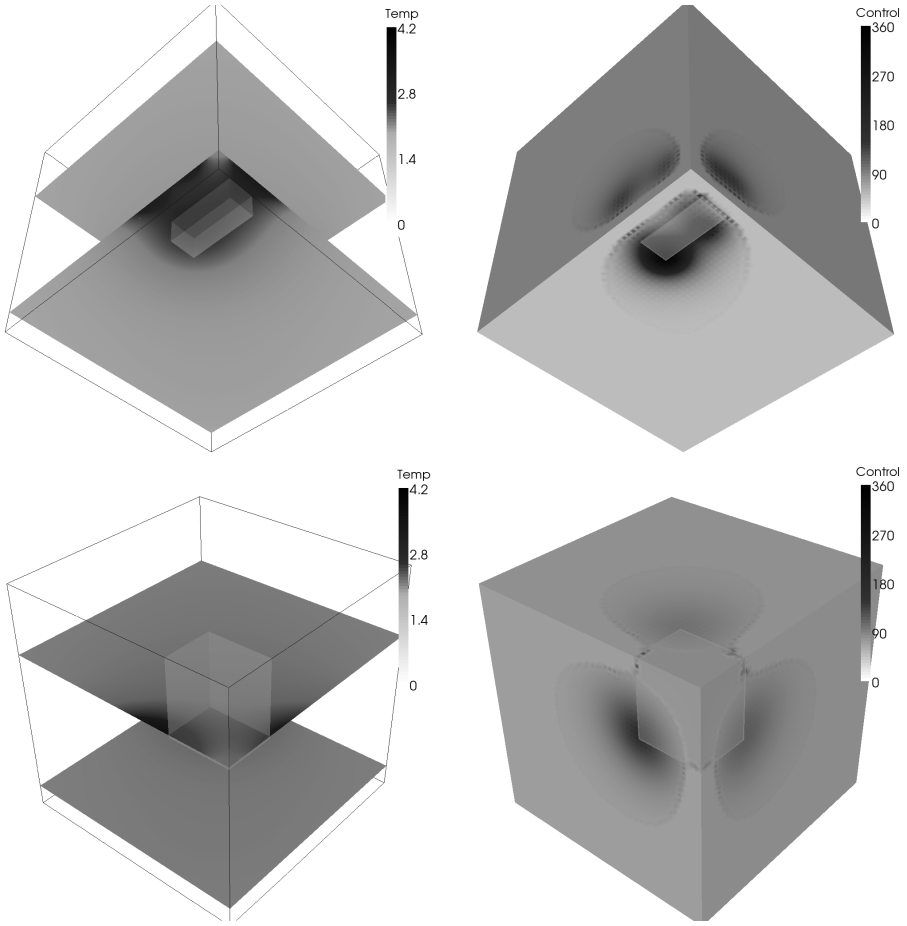

Fig. 4.1 Optimal state (left) and control (right) for the boundary control example. The regions Ω_1 (top) and Ω_2 (bottom) are visualized as a box. It is interesting to note that the corners of the regions Ω_1 and Ω_2 are heated most, instead of the inner part of its surface.

h	it	$f(x_*)$	CPUs	CPUs/it
0.05	29	40.6349	675.85	23.31
0.04	33	39.9458	2806.16	85.04
0.03	34	37.7909	16330.56	480.31
0.02	46	40.9115	304780.45	6625.66

Table 4.2 Performance measures for the default algorithm applied to the boundary control example.

h	it	$f(x_*)$	CPUs	CPUs/it	speedup
0.05	33	40.6349	374.17	11.34	1.81
0.04	33	39.9458	646.77	19.60	4.34
0.03	37	37.7909	4495.83	121.51	3.63
0.02	47	40.9115	38824.33	826.05	7.85

Table 4.3 Performance measures for the inexact algorithm applied to the boundary control example.

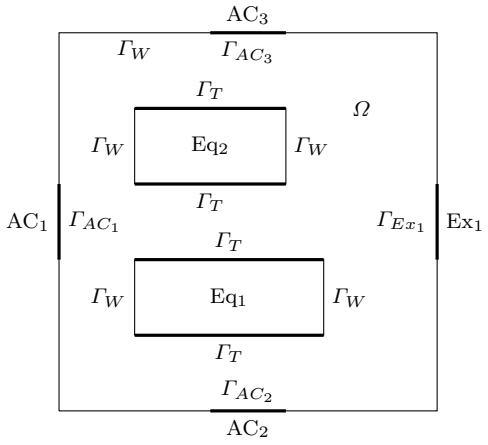

Fig. 4.2 Illustration of the geometry of the server room cooling model projected onto the $x_3 = 0$ axis. Cool air is pumped into the room via the AC units on the boundary in order to cool the hot surfaces of the equipment Γ_T. Air flows out of the room via the exhaust at Γ_{Ex_1}.

The tables list the average CPU time per iteration, but it should be noted that the step computation requires considerably more time towards the end of the optimization procedure than at the beginning. Taking the $h = 0.02$ case as an example, in the first 22 `IPOPT` iterations the preconditioners (each computed in less than one minute) have fill-in factors of at most 3 and SQMR requires only between 35 and 200 iterations, leading to times of less than 4 minutes for each step computation. However, in the last `IPOPT` iterations, the dropping tolerances have to be tightened (down to about $3{\cdot}10^{-4}$ and $3{\cdot}10^{-5}$, respectively). At the tightest level of these tolerances, the preconditioner (computed in up to 9 minutes) has a fill-in factor of almost 10 and still SQMR requires more than 1000 iterations, leading to times up to 35 minutes at this level. Even though our results demonstrate significant improvements due to the use of iterative linear system solvers, this illustrates that finding preconditioners that are less dependent on the conditioning of the saddle point matrix in (2.2) as μ approaches zero is still an area of active research (see, e.g., [1,4]).

4.3 Server room cooling

Our second example is motivated by the real-life problem of cooling computer equipment in a server room. In our simplified model, we assume that (cold) air is blown into the room from air conditioners (AC), and that (hot) air leaves the room at exhausts (Ex); see Figure 4.2. Inside the domain lies equipment with hot surfaces that need to be cooled by sufficient airflow passing alongside.

For simplicity, we suppose that air is incompressible, has no internal friction, and that all velocities are far below the speed of sound. Under these assumptions, we can model air velocity $y(x)$ as the gradient of a potential $\Phi(x)$ satisfying the Laplace equation

$$-\Delta\Phi = 0 \qquad \text{in } \Omega \tag{4.4}$$

for a domain $\Omega \subseteq \mathbb{R}^3$. Appropriate boundary conditions for the walls (and non-heat producing surfaces of the equipment) Γ_W, cold air inlets Γ_{AC_i}, exhausts Γ_{Ex_i}, and heat producing surfaces of the equipment Γ_T, respectively, are

$$\frac{\partial \Phi}{\partial n} = 0 \qquad \text{on } \Gamma_W \tag{4.5a}$$
$$\frac{\partial \Phi}{\partial n} = -u_{AC_i} \Psi_{\Gamma_{AC_i}} \qquad \text{on } \Gamma_{AC_i} \tag{4.5b}$$
$$\frac{\partial \Phi}{\partial n} = +u_{Ex_i} \Psi_{\Gamma_{Ex_i}} \qquad \text{on } \Gamma_{Ex_i} \tag{4.5c}$$
$$\frac{\partial \Phi}{\partial n} = 0 \qquad \text{on } \Gamma_T; \tag{4.5d}$$

see also Figure 4.2. Here, $\frac{\partial \Phi}{\partial n}$ denotes the outward-pointing normal derivative of the potential, and $\Psi_{\Gamma_{AC_i}}(x)$ $(i = 1, \ldots, N_{AC})$ and $\Psi_{\Gamma_{Ex_i}}(x)$ $(i = 1, \ldots, N_{Ex})$ define airflow velocity profiles on the surfaces of the air conditioners and exhausts, respectively. Similarly, $u_{AC_i} \in \mathbb{R}$ and $u_{Ex_i} \in \mathbb{R}$ denote control parameters for the maximal flow rates at these air inlets and outlets. The weak formulation of (4.4) with (4.5) is to find $\Phi \in H^1(\Omega)$ such that

$$\begin{aligned} 0 = & -\int_\Omega \Delta \Phi v \, dx \\ = & \int_\Omega \nabla \Phi \cdot \nabla v \, dx + \sum_{i=1}^{N_{AC}} \int_{\Gamma_{AC_i}} u_{AC_i} \Psi_{\Gamma_{AC_i}} v \, da \\ & - \sum_{i=1}^{N_{Ex}} \int_{\Gamma_{Ex_i}} u_{Ex_i} \Psi_{\Gamma_{Ex_i}} v \, da \qquad \forall v \in H^1(\Omega). \end{aligned} \tag{4.6}$$

It is important to note that (4.6) has a solution only if the controls satisfy the mass balance equation

$$\sum_{i=1}^{N_{AC}} \int_{\Gamma_{AC_i}} u_{AC_i} \Psi_{\Gamma_{AC_i}} da - \sum_{i=1}^{N_{Ex}} \int_{\Gamma_{Ex_i}} u_{Ex_i} \Psi_{\Gamma_{Ex_i}} da = 0, \tag{4.7}$$

and in that case (4.6) only determines the potential $\Phi \in H^1(\Omega)$ up to an additive constant. Therefore, a normalization condition will be introduced below.

As a constraint, we require that the air speed at the heat-producing surfaces has a minimum velocity so that heat is carried away. More precisely, recalling that the velocity is the gradient of the potential function Φ, we impose a point-wise state constraint

$$\|\nabla \Phi(x)\|_2^2 \geq y_{\min}^2 \qquad \text{for all } x \in \Gamma_T \tag{4.8}$$

with a constant $y_{\min} > 0$.

To obtain the discretized problem, we generate an irregular mesh of tetrahedrons, each with maximal volume h^3, again choose a finite-dimensional subset $V^h \subseteq H^1(\Omega)$ with a basis $\{\varphi_i\}_{i=1,\ldots,n_h}$, and express the finite-dimensional approximation Φ_h of $\Phi = \sum_i \phi^{(i)} \varphi_i$ with coefficients $\phi \in \mathbb{R}^{n_h}$. Defining $u = (u_{AC}, u_{Ex})$ as the vector consisting of all control parameters, the discretized PDE (4.6) then becomes

$$A\phi - Bu = 0,$$

where A denotes the stiffness matrix $A^{(i,j)} = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j dx$, and $B = [B_{AC}\ B_{Ex}]$ implements the boundary conditions with $B_{AC}^{(i,j)} = -\int_{\Gamma_{AC_j}} \Psi_{\Gamma_{AC_j}} \varphi_i \, da$ and $B_{Ex}^{(i,j)} = \int_{\Gamma_{Ex_j}} \Psi_{\Gamma_{AC_j}} \varphi_i \, da$.

Thus, the finite-dimensional optimization problem is

$$\min_{\phi_i, u_i, \bar{u}} \ \sum \beta_j u_{AC_j}$$

subject to

$$A\phi - Bu + \gamma e\bar{u} = 0 \tag{4.9a}$$

$$\gamma e^T \phi - \bar{\gamma}\bar{u} = 0 \tag{4.9b}$$

$$e^T Bu = 0 \tag{4.9c}$$

$$\int_{\Gamma_e} \nabla\phi(x) \cdot \nabla\phi(x) \, da - y_{\min}^2 \left(\int_{\Gamma_e} da \right) \geq 0 \text{ for } \Gamma_e \subseteq \Gamma_T \tag{4.9d}$$

$$u \geq 0 \tag{4.9e}$$

with weights $\beta_i > 0$ in the objective function, and $e = (1, \ldots, 1)^T \in \mathbb{R}^{n_h}$. Here, (4.9c) is a compact way of writing (4.7), and (4.9d) is the discretized version of (4.8), which is posed for all element faces Γ_e contained in a heat producing surface Γ_T. Note that the constraint (4.9d) is nonlinear and nonconvex. Again, in our implementation of the above problem, we scaled the constraints (4.9a) and (4.9d) by factors $10^{-2}/h$ and $10^{-1}/h$, respectively, to ensure that the gradients of those functions do not vanish as $h \to 0$.

To overcome the ill-posedness of the PDE, an auxiliary variable $\bar{u} \in \mathbb{R}$ has been added to the problem statement. Eqn. (4.9a) includes the discretized PDE, where the term $\gamma e\bar{u}$ acts as a constant virtual source or sink all over Ω. Since we impose the mass conservation in (4.9c) explicitly, this term eventually yields $\bar{u} = 0$. Furthermore, an integral-type equation is imposed in (4.9b). Indeed, $e^T\phi$ can be understood as a discretization of $\int_\Omega \Phi d\mu$ for some measure μ depending on the finite-element discretization and is eventually set to zero in (4.9b) since $\bar{u} = 0$, therefore normalizing the velocity potential Φ. Arguing alternatively from a linear-algebra point-of-view, while the linear system $A\phi = b$ determining the state variables ϕ is singular with e being an eigenvector corresponding to the eigenvalue 0, it can be shown that the linear system

$$\begin{bmatrix} A & \gamma e \\ \gamma e^T & -\bar{\gamma} \end{bmatrix} \begin{bmatrix} \phi \\ u \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

is non-singular and provides a solution satisfying $A\phi = b$.

For our experiments we choose $\beta_i = 1$, $\gamma = 1$, $\bar{\gamma} = 10^8$, $y_{\min} = 1$, $\Gamma_{AC_1} = \{0\} \times [0.4, 0.6] \times [0.2, 0.4]$, $\Gamma_{AC_2} = [0.4, 0.6] \times \{0\} \times [0.2, 0.4]$, $\Gamma_{AC_3} = [0.4, 0.6] \times \{1\} \times [0.2, 0.4]$, and $\Gamma_{Ex_1} = \{1\} \times [0.4, 0.6] \times [0.6, 0.8]$. The equipment is placed so that $\Omega_{Eq_1} = [0.2, 0.7] \times [0.2, 0.4] \times [0, 0.8]$ and $\Omega_{Eq_2} = [0.2, 0.6] \times [0.6, 0.8] \times [0, 0.8]$ with the remaining boundary components Γ_T and Γ_W defined accordingly as illustrated in Figure 4.2. The airflows at the inlets and outlets are assumed to have quadratic profiles, e.g., on $\Gamma_{AC_i} = \{a^{(1)}\} \times [a^{(2)}, b^{(2)}] \times [a^{(3)}, b^{(3)}]$, we choose

$$\Psi_\Gamma(x) = \frac{4(x^{(2)} - a^{(2)})(b^{(2)} - x^{(2)})}{(b^{(2)} - a^{(2)})^2} \cdot \frac{4(x^{(3)} - a^{(3)})(b^{(3)} - x^{(3)})}{(b^{(3)} - a^{(3)})^2}.$$

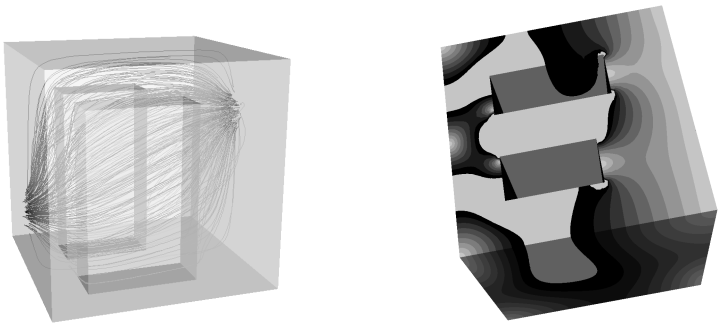

Fig. 4.3 Optimal solution of the server room cooling optimization example. On the left, we see the streamlines of the airflow, going from the main AC on the left to the exhaust on the right. On the right, we have a bottom view of the domain Ω, where the colors have been chosen to be dark if the air velocity is close to the threshold $y_{\min} = 1$. One can clearly see a region at the wall of the larger piece of equipment, at which the velocity is close to critical, indicating the location of the active constraints (4.9d) in Γ_T.

h	#var	#bds	#eq	#ineq
0.04	38582	4	38579	869
0.03	88398	4	88395	1528
0.02	285510	4	285507	3409
0.015	663886	4	663883	6110

Table 4.4 Problem sizes for instances of the server room cooling example.

Due to the nooks in Ω created by the equipment, numerical experiments with linear finite elements showed only linear L^2-convergence of the PDE solution as $h \to 0$. However, since (4.8) involves the gradient of the state variable, superlinear convergence is crucial. Thus, we have chosen quadratic finite-elements and observed quadratic convergence for the PDE solution. Specifically, for three choices of the mesh size parameter, $h = 0.2, 0.1, 0.05$, we computed the state variables from (4.9a)–(4.9b) for fixed values of the control parameters. Then we refined the mesh, corresponding to a value of $h/2$, and recomputed the state variables. The L^2-differences for the refined and original mesh were calculated as $3.87 \cdot 10^{-3}$, $1.04 \cdot 10^{-3}$ and $2.50 \cdot 10^{-4}$. Thus we observed factors of 3.7 and 4.2 for a bisection in h, which indicates quadratic convergence.

Table 4.4 shows problem size information for various instances of this problem. As starting point for our experiments, we calculated the solution of (4.9a)–(4.9c) for $u_{AC_i} = 20$. Tables 4.5 and 4.6 provide performance measures for the default `IPOPT` algorithm and for our implementation, respectively, where now we break down the optimal objective values into those for the control variables u_{AC_i} for each of the three air conditioners. Also here we see a clear reduction in computation time achieved by using the inexact algorithm, without a loss in solution accuracy. Specifically, the computation time for the largest instance with more than $600,000$ variables was reduced from more than 6 days to 8.2 hours, a speedup by a factor of 17.89. Figure 4.3 shows the optimal solution for the finest discretization.

h	it	$f(x_*)$	AC_1	AC_2	AC_3	CPUs	CPUs/it
0.04	23	15.4372	15.102	0.335	0.0	1019.16	44.31
0.03	21	15.5283	15.235	0.293	0.0	4511.48	214.83
0.02	33	15.5694	15.311	0.258	0.0	69427.33	2103.86
0.015	32	15.6509	15.428	0.223	0.0	528320.22	16510.01

Table 4.5 Performance measures for the default algorithm applied to the server room cooling example.

h	it	$f(x_*)$	AC_1	AC_2	AC_3	CPUs	CPUs/it	speedup
0.04	20	15.4372	15.102	0.335	0.0	622.31	31.12	1.64
0.03	24	15.5283	15.235	0.293	0.0	1710.30	71.26	2.64
0.02	28	15.5694	15.311	0.258	0.0	10008.50	357.45	6.94
0.015	27	15.6509	15.428	0.223	0.0	29526.53	1093.58	17.89

Table 4.6 Performance measures for the inexact algorithm applied to the server room cooling example.

In this example, the default settings for the preconditioner thresholds were sufficient in each iteration, so that no tightening occurred. For the $h = 0.015$ case, the computation time for the preconditioner ranged from 164 to 234 seconds (with an average of 204 seconds), the number of SQMR iterations was 204–1129 with an average of 396, and the time spent in SQMR ranged from 365 to 2018 seconds (with an average of 719 seconds). While there is some variation, we did not observe such a clear degeneration of computation time per iteration towards the end of the optimization as we saw for the example in §4.2.

5 Conclusion and Final Remarks

We have presented a detailed description of an implementation of a primal-dual interior-point method for large-scale nonconvex optimization where the search directions are computed inexactly by means of an iterative linear system solver. Ideally, the algorithm computes a search direction through the inexact solution of a single linear system (as in [14]). However, when appropriate, it falls back on the step decomposition strategy proposed in [20] so that, overall, the strong global convergence properties presented in [20] are attained. Numerical experiments on a large set of test problems and on two PDE-constrained optimization problems have also been presented. These results demonstrate the robustness of the approach and illustrate the significant speedup our algorithm attains when compared to an algorithm based on direct factorizations of the primal-dual system matrices.

As mentioned at the end of §4.2, we have observed a decrease in effectiveness of the preconditioner as the barrier parameter approaches zero. It remains a subject of future research to explore whether the specific structure of the ill-conditioning caused by the log-barrier terms can be handled efficiently within the incomplete-factorization-based preconditioner described in §3.4.

In our experiments with PDE-constrained optimization problems, we did not use any tailored preconditioners that take into account the spectral properties of the discrete differential operators which appear as a submatrix of the constraint Jacobian A_k. Note, however, that their use is not entirely straightforward. The challenge lies in constructing a preconditioner for the *entire* iteration matrix (2.9)

that efficiently employs such a tailored PDE preconditioner. Because this matrix includes the Hessian of the barrier term, it becomes increasingly ill-conditioned as $\mu \to 0$, and it is an open question whether it can be preconditioned efficiently.

Acknowledgments

The authors thank Michael Henderson, Vanessa Lopez, and Ulisses Mello for suggesting the server room airflow example, and Madam Sathe for discussions regarding the preconditioner. They also thank the anonymous referees whose comments and suggestions led to significant improvements. Johannes Huber gratefully acknowledges financial support from IBM during his summer internship at the IBM Watson Research Center.

References

1. G. Al-Jeiroudi, J. Gondzio, and J. Hall. Preconditioning indefinite systems in interior point methods for large scale linear optimisation. *Optimization Methods and Software*, 23(3):345–364, 2008.
2. V. Arnautu and P. Neittaanmaki. *Optimal Control from Theory to Computer Programs*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.
3. S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.
4. M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, (14):1–137, 2005.
5. J. T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Advances in Design and Control. SIAM, Philadelphia, PA, USA, 2001.
6. L. T. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. Van Bloemen Waanders. *Real-Time PDE-Constrained Optimization*. Computational Science and Engineering. SIAM, Philadelphia, PA, 2007.
7. L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. Van Bloemen Waanders, editors. *Large-Scale PDE-Constrained Optimization*. Lecture Notes in Computational Science and Engineering. Springer-Verlag, Berlin, Heidelberg, New York, 2003.
8. G. Biros and O. Ghattas. Inexactness Issues in the Lagrange-Newton-Krylov-Schur Method for PDE-constrained Optimization. In L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. Van Bloemen Waanders, editors, *Large-Scale PDE-Constrained Optimization*, pages 93–114, New York, NY, USA, 2003. Springer.
9. G. Biros and O. Ghattas. Parallel Lagrange-Newton-Krylov-Schur Methods for PDE-Constrained Optimization. Part I: The Krylov-Schur Solver. *SIAM Journal on Scientific Computing*, 27(2):687–713, 2005.
10. G. Biros and O. Ghattas. Parallel Lagrange-Newton-Krylov-Schur Methods for PDE-Constrained Optimization. Part II: The Lagrange-Newton Solver and Its Application to Optimal Control of Steady Viscous Flows. *SIAM Journal on Scientific Computing*, 27(2):714–739, 2005.
11. M. Bollhöfer and Y. Saad. Multilevel Preconditioners Constructed from Inverse-Based ILUs. *SIAM Journal on Scientific Computing*, 27(5):1627–1650, 2006.
12. A. Borzi and V. Schulz. Multigrid Methods for PDE Optimization. *SIAM Review*, 51(2):361–395, 2009.
13. R. H. Byrd, F. E. Curtis, and J. Nocedal. An Inexact SQP Method for Equality Constrained Optimization. *SIAM Journal on Optimization*, 19(1):351–369, 2008.
14. R. H. Byrd, F. E. Curtis, and J. Nocedal. An Inexact Newton Method for Nonconvex Equality Constrained Optimization. *Mathematical Programming*, 122(2):273–299, 2010.
15. R. H. Byrd, J.-Ch. Gilbert, and J. Nocedal. A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming. *Mathematical Programming*, 89(1):149–185, 2000.

16. R. H. Byrd, M. E. Hribar, and J. Nocedal. An Interior Point Algorithm for Large-Scale Nonlinear Programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.
17. A.K. Cline, Cleve B. Moler, G.W. Stewart, and J.H. Wilkinson. An estimate for the condition number of a matrix. *SIAM J. Numer. Anal.*, 16:368–375, 1979.
18. F. E. Curtis and J. Nocedal. Flexible Penalty Functions for Nonlinear Constrained Optimization. *IMA Journal of Numerical Analysis*, 28(4):749–769, 2008.
19. F. E. Curtis, J. Nocedal, and A. Wächter. A Matrix-Free Algorithm for Equality Constrained Optimization Problems with Rank Deficient Jacobians. *SIAM Journal on Optimization*, 20(3):1224–1249, 2009.
20. F. E. Curtis, O. Schenk, and A. Wächter. An Interior-Point Algorithm for Nonlinear Optimization with Inexact Step Computations. *SIAM Journal on Scientific Computing*, 32(6):3447–3475, 2010.
21. R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton Methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.
22. M. Fisher, J. Nocedal, Y. Trémolet, and S. J. Wright. Data Assimilation in Weather Forecasting: A Case Study in PDE-Constrained Optimization. *Optimization and Engineering*, 10(3):409–426, 2009.
23. R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Brooks/Cole, 2002.
24. R. W. Freund. Preconditioning of Symmetric, but Highly Indefinite Linear Systems. In A. Sydow, editor, *15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, pages 551–556, Berlin, 1997. Wissenschaft & Technik.
25. N. I. M. Gould, I. Bongartz, A. R. Conn, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.
26. N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr and SifDec: A Constrained and Unconstrained Testing Environment, Revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.
27. E. Haber and U. M. Ascher. Preconditioned All-at-Once Methods for Large, Sparse Parameter Estimation Problems. *Inverse Problems*, 17:1847–1864, 2001.
28. M. Heinkenschloss and D. Ridzal. An Inexact Trust-Region SQP Method with Applications to PDE-Constrained Optimization. In K. Kunisch, G. Of, and O. Steinbach, editors, *Numerical Mathematics and Advanced Applications: Proceedings of ENUMATH 2007, the 7th European Conference on Numerical Mathematics and Advanced Applications, Graz, Austria*, pages 613–620. Springer, 2008.
29. M. Heinkenschloss and L. N. Vicente. Analysis of Inexact Trust-Region SQP Algorithms. *SIAM Journal on Optimization*, 12(2):283–302, 2002.
30. M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE Constraints*, volume 23 of *Mathematical Modeling: Theory and Applications.* Springer, Dordrecht, Netherlands, 2009.
31. H. Jäger and E. W. Sachs. Global Convergence of Inexact Reduced SQP Methods. *Optimization Methods and Software*, 7(2):83–110, 1997.
32. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.
33. B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006.
34. D. E. Kirk. *Optimal Control Theory: An Introduction.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1970.
35. R. P. Pawlowski, J. P. Simonis, H. F. Walker, and J. N. Shadid. Inexact Newton Dogleg Methods. *SIAM Journal on Numerical Analysis*, 46(4):2112–2132, 2008.
36. M. J. D. Powell. A Hybrid Method for Nonlinear Equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114, London, 1970. Gordon and Breach.
37. E. E. Prudencio, R. H. Byrd, and X.-C. Cai. Parallel Full Space SQP Lagrange-Newton-Krylov-Schwarz Algorithms for PDE-Constrained Optimization Problems. *SIAM Journal on Scientific Computing*, 27(4):1305–1328, 2005.
38. D. Ridzal. *Trust-Region SQP Methods with Inexact Linear System Solves for Large-Scale Optimization.* PhD thesis, Rice University, 2006.
39. O. Schenk, M. Bollhöfer, and R. A. Roemer. On Large Scale Diagonalization Techniques for the Anderson Model of Localization. *SIAM Journal on Scientific Computing*, 28(3):963–983, 2006.

40. O. Schenk, M. Bollhöfer, and A. Römer. On Large-Scale Diagonalization Techniques for the Anderson Model of Localization. *SIAM Review*, 50(1):91–112, 2008.
41. O. Schenk and K. Gärtner. On Fast Factorization Pivoting Methods for Sparse Symmetric Indefinite Systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.
42. O. Schenk, A. Wächter, and M. Hagemann. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Comput. Optim. Appl.*, 36(2-3):321–341, April 2007.
43. O. Schenk, A. Wächter, and M. Weiser. Inertia Revealing Preconditioning for Large-Scale Nonconvex Constrained Optimization. *SIAM Journal on Scientific Computing*, 31(2):939–960, 2008.
44. T. Steihaug. The Conjugate Gradient Method and Trust Regions in Large Scale Optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.
45. F. Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods, and Applications*, volume 112. American Mathematical Society, 2010.
46. S. Ulbrich. Generalized SQP-Methods with "Parareal" Time-Domain Decomposition for Time-Dependent PDE-Constrained Optimization. In L. T. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, editors, *Real-Time PDE-Constrained Optimization*, pages 145–168, Philadelphia, PA, USA, 2007. SIAM.
47. A. Wächter and L. T. Biegler. Failure of Global Convergence for a Class of Interior Point Methods for Nonlinear Programming. *Mathematical Programming*, 88(3):565–574, 2000.
48. A. Wächter and L. T. Biegler. Line Search Filter Methods for Nonlinear Programming: Motivation and Global Convergence. *SIAM Journal on Optimization*, 16:1–31, 2005.
49. A. Wächter and L. T. Biegler. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, 106(1):25–57, 2006.
50. D. P. Young, W. P. Huffman, R. G. Melvin, C. L. Hilmes, and F. T. Johnson. Nonlinear Elimination in Aerodynamic Analysis and Design Optimization. In L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. Van Bloemen Waanders, editors, *Large-Scale PDE-Constrained Optimization*, pages 17–44, New York, NY, USA, 2003. Springer.