Science & Technology Facilities Council

Preprint

RAL-P-2014-011

Adaptive augmented Lagrangian methods algorithms and practical numerical experience

Curtis FE, Gould NIM, Jiang H, Robinson DP

August 2014

Submitted for publication in Optimization Methods and Software

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

Adaptive augmented Lagrangian methods: algorithms and practical numerical experience

Frank E. Curtis,[1,2] Nicholas I. M. Gould,[3,4], Hao Jiang[5,6] and Daniel P. Robinson[5,6]

ABSTRACT

In this paper, we consider augmented Lagrangian (AL) algorithms for solving large-scale nonlinear optimization problems that execute adaptive strategies for updating the penalty parameter. Our work is motivated by the recently proposed adaptive AL trust region method by Curtis, Jiang, and Robinson [Math. Prog., DOI: 10.1007/s10107-014-0784-y, 2013]. The first focal point of this paper is a new variant of the approach that employs a line search rather than a trust region strategy, where a critical algorithmic feature for the line search strategy is the use of convexified piecewise quadratic models of the AL function for computing the search directions. We prove global convergence guarantees for our line search algorithm that are on par with those for the previously proposed trust region method. A second focal point of this paper is the practical performance of the line search and trust region algorithm variants in MATLAB software, as well as that of an adaptive penalty parameter updating strategy incorporated into the LANCELOT software. We test these methods on problems from the CUTEST and COPS collections, as well as on challenging test problems related to optimal power flow. Our numerical experience suggests that the adaptive algorithms outperform traditional AL methods in terms of efficiency and reliability. As with traditional AL algorithms, the adaptive methods are matrix-free and thus represent a viable option for solving extreme-scale problems.

---

[1] Department of Industrial and Systems Engineering, Lehigh University,
Harold S. Mohler Laboratory, 200 West Packer Avenue Bethlehem, PA 18015-1582, USA.
Email : frank.e.curtis@gmail.com .

[2] This work was supported by U.S. Department of Energy grant DE–SC0010615.

[3] Scientific Computing Department, Rutherford Appleton Laboratory,
Chilton, Oxfordshire, OX11 0QX, England, EU. Email: nick.gould@stfc.ac.uk .
Current reports available from "http://www.numerical.rl.ac.uk/people/nimg/pubs.html".

[4] This work was supported by the EPSRC grant EP/I013067/1.

[5] Department of Applied Mathematics and Statistics, Johns Hopkins University,
100 Whitehead Hall, 3400 N. Charles Street, Baltimore, MD 21218-2682, USA.
Email : hjiang13@jhu.edu , daniel.p.robinson@gmail.com .

[6] This work was supported by the U.S. National Science Foundation grant DMS–1217153.

Scientific Computing Department
Rutherford Appleton Laboratory
Oxfordshire OX11 0QX

August 22, 2014

1 Introduction

Augmented Lagrangian (AL) methods [25, 32] have recently regained popularity due to growing interests in solving extreme-scale nonlinear optimization problems. These methods are attractive in such settings as they can be implemented matrix-free [2, 3, 11, 28] and have global and local convergence guarantees under relatively weak assumptions [18, 26]. Furthermore, certain variants of AL methods [20, 21] have proved to be very efficient for solving certain structured problems [6, 33, 35].

A new AL trust region method was recently proposed and analyzed in [14]. The novel feature of that algorithm is an adaptive strategy for updating the penalty parameter inspired by techniques for performing such updates in the context of exact penalty methods [7, 8, 29]. This feature is designed to overcome a potentially serious drawback of traditional AL methods, which is that they may be ineffective during some (early) iterations due to poor choices of the penalty parameter and/or Lagrange multiplier estimates. In such situations, the poor choices of these quantities may lead to little or no improvement in the primal space and, in fact, the iterates may diverge from even a well-chosen initial iterate. The key idea for avoiding this behavior in the algorithm proposed in [14] is to adaptively update the penalty parameter *during* the step computation in order to ensure that the trial step yields a sufficiently large reduction in linearized constraint violation, thus *steering* the optimization process steadily toward constraint satisfaction.

The contributions of this paper are two-fold. First, we present an AL line search method based on the same framework employed for the trust region method in [14]. The main difference between our new approach and that in [14], besides the differences inherent in using line searches instead of a trust region strategy, is that we utilize a convexified piecewise quadratic model of the AL function to compute the search direction in each iteration. With this modification, we prove that our line search method achieves global convergence guarantees on par with those proved for the trust region method in [14]. The second contribution of this paper is that we perform extensive numerical experiments with a MATLAB implementation of the adaptive algorithms (i.e., both line search and trust region variants) and an implementation of an adaptive penalty parameter updating strategy in the LANCELOT software [12]. We test these implementations on problems from the CUTEST [22] and COPS [5] collections, as well as on test problems related to optimal power flow [36]. Our results indicate that our adaptive algorithms outperform traditional AL methods in terms of efficiency and reliability.

The remainder of the paper is organized as follows. In §2, we present our adaptive AL line search method and state convergence results. Details about these results, which draw from those in [14], can be found in Appendices 4 and A.2 with further details in [15]. We then provide numerical results in §3 to illustrate the effectiveness of our implementations of our adaptive AL algorithms. We give conclusions in §4.

Notation. We often drop function arguments once a function is defined. We also use a subscript on a function name to denote its value corresponding to algorithmic quantities using the same subscript. For example, for a function $f : \mathbb{R}^n \to \mathbb{R}$, if x_k is the value for the variable x during iteration k of an algorithm, then $f_k := f(x_k)$. We also often use subscripts for constants to indicate the algorithmic quantity to which they correspond. For example, γ_μ denotes a parameter corresponding to the algorithmic quantity μ.

2 An Adaptive Augmented Lagrangian Line Search Algorithm

2.1 Preliminaries

We assume that all problems under our consideration are formulated as

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x)\ \text{ subject to }\ c(x)=0,\ \ l\le x\le u. \tag{1}$$

Here, we assume that the objective function $f:\mathbb{R}^n\to\mathbb{R}$ and constraint function $c:\mathbb{R}^n\to\mathbb{R}^m$ are twice continuously differentiable, and that the variable lower bound vector $l\in\mathbb{R}^n$ and upper bound vector $u\in\mathbb{R}^n$ satisfy $l\le u$. Our goal is to design an algorithm that will compute a first-order primal-dual stationary point for problem (1). However, in order for the algorithm to be suitable as a general-purpose approach, it should have mechanisms for terminating and providing useful information when an instance of (1) is (locally) infeasible. In such cases, we have designed our algorithm so that it transitions to finding a point that is infeasible with respect to (1), but is a first-order stationary point for the nonlinear feasibility problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ v(x)\ \text{ subject to }\ l\le x\le u, \tag{2}$$

where $v:\mathbb{R}^n\to\mathbb{R}$ is defined as $v(x)=\frac{1}{2}\|c(x)\|_2^2$.

As implied by the previous paragraph, our algorithm requires first-order stationarity conditions for problems (1) and (2), which can be stated in the following manner. First, introducing a Lagrange multiplier vector $y\in\mathbb{R}^m$, we define the Lagrangian for problem (1), call it $\ell:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}$, by

$$\ell(x,y)=f(x)-c(x)^Ty.$$

Then, defining the gradient of the objective function $g:\mathbb{R}^n\to\mathbb{R}^n$ by $g(x)=\nabla f(x)$, the Jacobian of the constraint functions $J:\mathbb{R}^n\to\mathbb{R}^{m\times n}$ by $J(x)=\nabla c(x)$, and the projection operator onto the bounds $P:\mathbb{R}^n\to\mathbb{R}^n$, component-wise for $i\in\{1,\dots,n\}$, by

$$[P(x)]_i=\begin{cases} l_i & \text{if } x_i\le l_i\\ u_i & \text{if } x_i\ge u_i\\ x_i & \text{otherwise}\end{cases}$$

we may introduce the primal-dual stationarity measure $F_{\text{L}}:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^n$ given by

$$F_{\text{L}}(x,y)=P(x-\nabla_x\ell(x,y))-x=P(x-(g(x)-J(x)^Ty))-x.$$

First-order primal-dual stationary points for (1) can then be characterized as zeros of the primal-dual stationarity measure $F_{\text{OPT}}:\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^{n+m}$ defined by stacking the stationarity measure F_{L} and the constraint function $-c$, i.e., a first-order primal-dual stationary point for (1) is any pair (x,y) with $l\le x\le u$ satisfying

$$0=F_{\text{OPT}}(x,y)=\begin{pmatrix}F_{\text{L}}(x,y)\\ -c(x)\end{pmatrix}=\begin{pmatrix}P(x-\nabla_x\ell(x,y))-x\\ \nabla_y\ell(x,y)\end{pmatrix}. \tag{3}$$

Similarly, a first-order primal stationary point for (2) is any x with $l\le x\le u$ satisfying

$$0=F_{\text{FEAS}}(x), \tag{4}$$

where $F_{\text{FEAS}} : \mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$F_{\text{FEAS}}(x) = P(x - \nabla_x v(x)) - x = P(x - J(x)^T c(x)) - x.$$

In particular, if $l \leq x \leq u$, $v(x) > 0$, and (4) holds, then x is an infeasible stationary point for problem (1).

Over the past decades, a variety of effective numerical methods have been proposed for solving large-scale bound-constrained optimization problems. Hence, the critical issue in solving problem (1) is how to handle the presence of the equality constraints. As in the wide variety of penalty methods that have been proposed, the strategy adopted by AL methods is to remove these constraints, but influence the algorithm to satisfy them through the addition of terms in the objective function. In this manner, problem (1) (or at least (2)) can be solved via a sequence of bound-constrained subproblems—thus allowing AL methods to exploit the methods that are available for subproblems of this type. Specifically, AL methods consider a sequence of subproblems in which the objective is a weighted sum of the Lagrangian ℓ and the constraint violation measure v. By scaling ℓ by a penalty parameter $\mu \geq 0$, each subproblem involves the minimization of a function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$, called the augmented Lagrangian (AL), defined by

$$\mathcal{L}(x, y, \mu) = \mu\ell(x, y) + v(x) = \mu(f(x) - c(x)^T y) + \tfrac{1}{2}\|c(x)\|_2^2.$$

Observe that the gradient of the AL with respect to x, evaluated at (x, y, μ), is given by

$$\nabla_x \mathcal{L}(x, y, \mu) = \mu\big(g(x) - J(x)^T \pi(x, y, \mu)\big),$$

where we define the function $\pi : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$ by

$$\pi(x, y, \mu) = y - \tfrac{1}{\mu} c(x).$$

Hence, each subproblem to be solved in an AL method has the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ \mathcal{L}(x, y, \mu) \ \text{ subject to } \ l \leq x \leq u. \tag{5}$$

Given a pair (y, μ), a first-order stationary point for problem (5) is any zero of the primal-dual stationarity measure $F_{\text{AL}} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$, defined similarly to F_{L} but with the Lagrangian replaced by the augmented Lagrangian; i.e., given (y, μ), a first-order stationary point for (5) is any x satisfying

$$0 = F_{\text{AL}}(x, y, \mu) = P(x - \nabla_x \mathcal{L}(x, y, \mu)) - x. \tag{6}$$

Given a pair (y, μ) with $\mu > 0$, a traditional AL method proceeds by (approximately) solving (5), which is to say that it finds a point, call it $x(y, \mu)$, that (approximately) satisfies (6). If the resulting pair $(x(y, \mu), y)$ is not a first-order primal-dual stationary point for (1), then the method would modify the Lagrange multiplier y or penalty parameter μ so that, hopefully, the solution of the subsequent subproblem (of the form (5)) yields a better primal-dual solution estimate for (1). The function π plays a critical role in this procedure. In particular, observe that if $c(x(y, \mu)) = 0$, then $\pi(x(y, \mu), y, \mu) = y$ and (6) would imply $F_{\text{OPT}}(x(y, \mu), y) = 0$, i.e., that $(x(y, \mu), y)$ is a first-order primal-dual stationary point for (1). Hence, if the constraint violation at $x(y, \mu)$ is sufficiently small, then a traditional AL method would set the new value of y as $\pi(x, y, \mu)$. Otherwise, if the constraint violation is not sufficiently small, then the penalty parameter is decreased to place a higher priority on reducing it during subsequent iterations.

2.2 Algorithm Description

Our AL line search algorithm is similar to the AL trust region method proposed in [14], except for two key differences: it executes line searches rather than using a trust region framework, and it employs a convexified piecewise quadratic model of the AL function for computing the search direction in each iteration. The main motivation for utilizing a convexified model is to ensure that each computed search direction is a direction of strict descent for the AL function from the current iterate, which is necessary to ensure the well-posedness of the line search. However, it should be noted that, practically speaking, the convexification of the model does not necessarily add any computational difficulties when computing each direction; see §3.1.1. Similar to the trust region method proposed in [14], a critical component of our algorithm is the adaptive strategy for updating the penalty parameter μ during the search direction computation. This is used to ensure steady progress—i.e., steer the algorithm—toward solving (1) (or at least (2)) by monitoring predicted improvements in linearized feasibility.

The central component of each iteration of our algorithm is the search direction computation. In our approach, this computation is performed based on local models of the constraint violation measure v and the AL function $\mathcal{L}$ at the current iterate, which at iteration k is given by (x_k, y_k, μ_k). The local models that we employ for these functions are, respectively, $q_v : \mathbb{R}^n \to \mathbb{R}$ and $\tilde{q} : \mathbb{R}^n \to \mathbb{R}$, defined as follows:

$$\begin{aligned} q_v(s;x) &= \tfrac{1}{2}\|c(x) + J(x)s\|_2^2 \\ \text{and } \tilde{q}(s;x,y,\mu) &= \mathcal{L}(x,y) + \nabla_x \mathcal{L}(x,y)^T s + \max\{\tfrac{1}{2} s^T(\mu \nabla^2_{xx}\ell(x,y) + J(x)^T J(x))s, 0\}. \end{aligned}$$

We note that q_v is a typical Gauss-Newton model of the constraint violation measure v, and $\tilde{q}$ is a convexification of a second-order approximation of the augmented Lagrangian. (We use the notation $\tilde{q}$ rather than simply q to distinguish between the model above and the second-order model—without the max—that appears extensively in [14].)

Our algorithm computes two types of steps during each iteration. The purpose of the first step, which we refer to as the steering step, is to gauge the progress towards linearized feasibility that may be achieved (locally) from the current iterate. This is done by (approximately) minimizing our model q_v of the constraint violation measure v within the bound constraints and a trust region. Then, a step of the second type is computed by (approximately) minimizing our model $\tilde{q}$ of the AL function $\mathcal{L}$ within the bound constraints and a trust region. If the reduction in the model q_v yielded by the latter step is sufficiently large—say, compared to that yielded by the steering step—then the algorithm proceeds using this step as the search direction. Otherwise, the penalty parameter may be reduced, in which case a step of the latter type is recomputed. This process repeats iteratively until a search direction is computed that yields a sufficiently large (or at least not too negative) reduction in q_v. As such, the iterate sequence is intended to make steady progress toward (or at least approximately maintain) constraint satisfaction throughout the optimization process, regardless of the initial penalty parameter value.

We now describe this process in more detail. During iteration k, the steering step r_k is computed via the optimization subproblem given by

$$\underset{r \in \mathbb{R}^n}{\text{minimize}} \; q_v(r; x_k) \;\; \text{subject to} \;\; l \leq x_k + r \leq u, \; \|r\|_2 \leq \theta_k, \tag{7}$$

where, for some constant $\delta > 0$, the trust region radius is defined to be

$$\theta_k := \delta \|F_{\text{FEAS}}(x_k)\|_2 \geq 0. \tag{8}$$

A consequence of this choice of trust region radius is that it forces the steering step to be smaller in norm as the iterates of the algorithm approach any stationary point of the constraint violation measure [34]. This prevents the steering step from being too large relative to the progress that can be made toward minimizing v. While (7) is a convex optimization problem for which there are efficient methods, in order to reduce computational expense our algorithm only requires r_k to be an approximate solution of (7). In particular, we merely require that r_k yields a reduction in q_v that is proportional to that yielded by the associated Cauchy step (see (13a) later on), which is defined to be

$$\overline{r}_k := \overline{r}(x_k, \theta_k) := P(x_k - \overline{\beta}_k J_k^T c_k) - x_k$$

for $\overline{\beta}_k := \overline{\beta}(x_k, \theta_k)$ such that, for some $\varepsilon_r \in (0, 1)$, the step $\overline{r}_k$ satisfies

$$\Delta q_v(\overline{r}_k; x_k) := q_v(0; x_k) - q_v(\overline{r}_k; x_k) \geq -\varepsilon_r \overline{r}_k^T J_k^T c_k \;\text{ and }\; \|\overline{r}_k\|_2 \leq \theta_k. \tag{9}$$

Appropriate values for $\overline{\beta}_k$ and $\overline{r}_k$—along with auxiliary nonnegative scalar quantities ε_k and Γ_k to be used in subsequent calculations in our method—are computed by Algorithm 1. The quantity $\Delta q_v(\overline{r}_k; x_k)$ representing the predicted reduction in constraint violation yielded by $\overline{r}_k$ is guaranteed to be positive at any x_k that is not a first-order stationary point for v subject to the bound constraints; see part (i) of Lemma A.4. We define a similar reduction $\Delta q_v(r_k; x_k)$ for the steering step r_k.

Algorithm 1 Cauchy step computation for the feasibility subproblem (7)

procedure CAUCHY_FEASIBILITY(x_k, θ_k)
  restrictions : $\theta_k \geq 0$.
  available constants : $\{\varepsilon_r, \gamma\} \subset (0, 1)$.
  Compute the smallest integer $l_k \geq 0$ satisfying $\|P(x_k - \gamma^{l_k} J_k^T c_k) - x_k\|_2 \leq \theta_k$.
  if $l_k > 0$ **then**
    Set $\Gamma_k \leftarrow \min\{2, \frac{1}{2}(1 + \|P(x_k - \gamma^{l_k - 1} J_k^T c_k) - x_k\|_2 / \theta_k)\}$.
  else
    Set $\Gamma_k \leftarrow 2$.
  end if
  Set $\overline{\beta}_k \leftarrow \gamma^{l_k}$, $\overline{r}_k \leftarrow P(x_k - \overline{\beta}_k J_k^T c_k) - x_k$, and $\varepsilon_k \leftarrow 0$.
  while $\overline{r}_k$ does not satisfy (9) **do**
    Set $\varepsilon_k \leftarrow \max(\varepsilon_k, -\Delta q_v(\overline{r}_k; x_k) / \overline{r}_k^T J_k^T c_k)$.
    Set $\overline{\beta}_k \leftarrow \gamma \overline{\beta}_k$ and $\overline{r}_k \leftarrow P(x_k - \overline{\beta}_k J_k^T c_k) - x_k$.
  end while
  return : $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k)$
end procedure

After computing a steering step r_k, we proceed to compute a trial step s_k via

$$\underset{s \in \mathbb{R}^n}{\text{minimize}}\; \tilde{q}(s; x_k, y_k, \mu_k) \;\text{ subject to }\; l \leq x_k + s \leq u,\; \|s\|_2 \leq \Theta_k, \tag{10}$$

where, given $\Gamma_k > 1$ from the output of Algorithm 1, we define the trust region radius

$$\Theta_k := \Theta(x_k, y_k, \mu_k, \Gamma_k) = \Gamma_k \delta \|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2 \geq 0. \tag{11}$$

As for the steering step, we allow inexactness in the solution of (10) by only requiring the step s_k to satisfy a Cauchy decrease condition (see (13b) later on), where the Cauchy step for problem (10) is

$$\bar{s}_k := \bar{s}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k) := P(x_k - \bar{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)) - x_k$$

for $\bar{\alpha}_k = \bar{\alpha}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$ such that, for $\varepsilon_k \geq 0$ returned from Algorithm 1, $\bar{s}_k$ yields

$$\begin{aligned} \Delta\tilde{q}(\bar{s}_k; x_k, y_k, \mu_k) &:= \tilde{q}(0; x_k, y_k, \mu_k) - \tilde{q}(\bar{s}_k; x_k, y_k, \mu_k) \\ &\geq -\frac{(\varepsilon_k + \varepsilon_r)}{2} \bar{s}_k^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k) \ \text{ and } \ \|\bar{s}_k\|_2 \leq \Theta_k. \end{aligned} \tag{12}$$

Algorithm 2 describes our procedure for computing $\bar{\alpha}_k$ and $\bar{s}_k$. (The importance of incorporating Γ_k in (11) and ε_k in (12) is revealed in the proofs of Lemmas A.2 and A.3; see [15].) The quantity $\Delta\tilde{q}(\bar{s}_k; x_k, y_k, \mu_k)$ representing the predicted reduction in $\mathcal{L}(\cdot, y_k, \mu_k)$ yielded by $\bar{s}_k$ is guaranteed to be positive at any x_k that is not a first-order stationary point for $\mathcal{L}(\cdot, y_k, \mu_k)$ subject to the bound constraints; see part (ii) of Lemma A.4. A similar quantity $\Delta\tilde{q}(s_k; x_k, y_k, \mu_k)$ is also used for the search direction s_k.

Algorithm 2 Cauchy step computation for the Augmented Lagrangian subproblem (10)

1: **procedure** CAUCHY_AL($x_k, y_k, \mu_k, \Theta_k, \varepsilon_k$)
2: **restrictions** : $\mu_k > 0$, $\Theta_k > 0$, and $\varepsilon_k \geq 0$.
3: **available constant** : $\gamma \in (0, 1)$.
4: Set $\bar{\alpha}_k \leftarrow 1$ and $\bar{s}_k \leftarrow P(x_k - \bar{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)) - x_k$.
5: **while** (12) is not satisfied **do**
6: Set $\bar{\alpha}_k \leftarrow \gamma \bar{\alpha}_k$ and $\bar{s}_k \leftarrow P(x_k - \bar{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)) - x_k$.
7: **end while**
8: **return** : $(\bar{\alpha}_k, \bar{s}_k)$
9: **end procedure**

Our complete algorithm is given as Algorithm 3 on page 8. In particular, the kth iteration proceeds as follows. Given the kth iterate tuple (x_k, y_k, μ_k), the algorithm first determines whether the first-order primal-dual stationarity conditions for (1) or the first-order stationarity condition for (2) are satisfied. If either is the case, then the algorithm terminates, but otherwise the method enters the **while** loop in line 13 to check for stationarity with respect to the AL function. This loop is guaranteed to terminate finitely; see Lemma A.1. Next, after computing appropriate trust region radii and Cauchy steps, the method enters a block for computing the steering step r_k and trial step s_k. Through the **while** loop on line 21, the overall goal of this block is to compute (approximate) solutions of subproblems (7) and (10) satisfying

$$\Delta\tilde{q}(s_k; x_k, y_k, \mu_k) \geq \kappa_1 \Delta\tilde{q}(\bar{s}_k; x_k, y_k, \mu_k) > 0, \quad l \leq x_k + s_k \leq u, \quad \|s_k\|_2 \leq \Theta_k, \tag{13a}$$

$$\Delta q_v(r_k; x_k) \geq \kappa_2 \Delta q_v(\bar{r}_k; x_k) \geq 0, \qquad l \leq x_k + r_k \leq u, \quad \|r_k\|_2 \leq \theta_k, \tag{13b}$$

$$\text{and} \ \ \Delta q_v(s_k; x_k) \geq \min\{\kappa_3 \Delta q_v(r_k; x_k), v_k - \tfrac{1}{2}(\kappa_t t_j)^2\}. \tag{13c}$$

In these conditions, the method employs user-provided constants $\{\kappa_1, \kappa_2, \kappa_3, \kappa_t\} \subset (0, 1)$ and the algorithmic quantity $t_j > 0$ representing the jth constraint violation target. It should be noted that, for sufficiently small $\mu > 0$, many approximate solutions to (7) and (10) satisfy (13), but for our purposes (see Theorem 2.2) it is sufficient that, for sufficiently small $\mu > 0$, they are at least satisfied by $r_k = \bar{r}_k$ and $s_k = \bar{s}_k$. A complete description of the motivations underlying conditions (13) can be found in [14, Section 3]. In short, (13a) and (13b) are Cauchy decrease conditions while (13c) ensures that the trial step predicts progress toward constraint satisfaction, or at least predicts that any increase in constraint violation is limited (when the right-hand side is negative).

With the search direction s_k in hand, the method proceeds to perform a backtracking line search along the strict descent direction s_k for $\mathcal{L}(\cdot, y_k, \mu_k)$ at x_k. Specifically, for a given $\gamma_\alpha \in (0, 1)$, the method computes the smallest integer $l \geq 0$ such that

$$\mathcal{L}(x_k + \gamma_\alpha^l s_k, y_k, \mu_k) \leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s \gamma_\alpha^l \Delta\widetilde{q}(s_k; x_k, y_k, \mu_k), \tag{14}$$

and then sets $\alpha_k \leftarrow \gamma_\alpha^l$ and $x_{k+1} \leftarrow x_k + \alpha_k s_k$. The remainder of the iteration is then composed of potential modifications of the Lagrange multiplier vector and target values for the accuracies in minimizing the constraint violation measure and AL function subject to the bound constraints. First, the method checks whether the constraint violation at the next primal iterate x_{k+1} is sufficiently small compared to the target $t_j > 0$. If this requirement is met, then a multiplier vector $\widehat{y}_{k+1}$ that satisfies

$$\|F_{\text{L}}(x_{k+1}, \widehat{y}_{k+1})\|_2 \leq \min\left\{\|F_{\text{L}}(x_{k+1}, y_k)\|_2, \|F_{\text{L}}(x_{k+1}, \pi(x_{k+1}, y_k, \mu_k))\|_2\right\} \tag{15}$$

is computed. Two obvious potential choices for $\widehat{y}_{k+1}$ are y_k and $\pi(x_{k+1}, y_k, \mu_k)$, but another viable candidate would be an approximate least-squares multiplier estimate (which may be computed via a linearly constrained optimization subproblem). The method then checks if either $\|F_{\text{L}}(x_{k+1}, \widehat{y}_{k+1})\|_2$ or $\|F_{\text{AL}}(x_{k+1}, y_k, \mu_k)\|_2$ is sufficiently small with respect to the target value $T_j > 0$. If so, then new target values $t_{j+1} < t_j$ and $T_{j+1} < T_j$ are set, $Y_{j+1} \geq Y_j$ is chosen, and a new Lagrange multiplier vector is set as

$$y_{k+1} \leftarrow (1 - \alpha_y) y_k + \alpha_y \widehat{y}_{k+1}, \tag{16}$$

where α_y is the largest value in $[0, 1]$ such that

$$\|(1 - \alpha_y) y_k + \alpha_y \widehat{y}_{k+1}\|_2 \leq Y_{j+1}. \tag{17}$$

This updating procedure is well-defined since the choice $\alpha_y \leftarrow 0$ results in $y_{k+1} \leftarrow y_k$, for which (17) is satisfied since $\|y_k\|_2 \leq Y_j \leq Y_{j+1}$. If either line 28 or line 30 in Algorithm 3 tests false, then the method simply sets $y_{k+1} \leftarrow y_k$. We note that unlike more traditional augmented Lagrangian approaches [2, 11], the penalty parameter is not adjusted on the basis of a test like that on line 28, but instead relies on our steering procedure. Moreover, in our approach we decrease the target values at a linear rate for simplicity, but more sophisticated approaches may be used [11].

2.3 Well-posedness and global convergence

In this section, we state two vital results, namely that Algorithm 3 is well posed, and that limit points of the iterate sequence have desirable properties. Vital components of these results

Algorithm 3 Adaptive Augmented Lagrangian Line Search Algorithm

Choose $\{\gamma, \gamma_\mu, \gamma_\alpha, \gamma_t, \gamma_T, \kappa_1, \kappa_2, \kappa_3, \varepsilon_r, \kappa_t, \eta_s, \eta_{vs}\} \subset (0,1)$ such that $\eta_{vs} \geq \eta_s$.
Choose $\{\delta, \epsilon, Y\} \subset (0, \infty)$.
Choose an initial primal-dual pair (x_0, y_0).
Choose $\{\mu_0, t_0, t_1, T_1, Y_1\} \subset (0, \infty)$ such that $Y_1 \geq \max\{Y, \|y_0\|_2\}$.
Set $k \leftarrow 0$, $k_0 \leftarrow 0$, and $j \leftarrow 1$.
loop
  if $F_{\text{OPT}}(x_k, y_k) = 0$, **then**
    return the first-order stationary solution (x_k, y_k).
  end if
  if $\|c_k\|_2 > 0$ and $F_{\text{FEAS}}(x_k) = 0$, **then**
    return the infeasible stationary point x_k.
  end if
  while $F_{\text{AL}}(x_k, y_k, \mu_k) = 0$, **do**
    Set $\mu_k \leftarrow \gamma_\mu \mu_k$.
  end while
  Set θ_k by (8).
  Use Algorithm 1 to compute $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) \leftarrow$ CAUCHY_FEASIBILITY(x_k, θ_k).
  Set Θ_k by (11).
  Use Algorithm 2 to compute $(\overline{\alpha}_k, \overline{s}_k) \leftarrow$ CAUCHY_AL$(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
  Compute approximate solutions r_k to (7) and s_k to (10) that satisfy (13a)–(13b).
  while (13c) is not satisfied or $F_{\text{AL}}(x_k, y_k, \mu_k) = 0$, **do**
    Set $\mu_k \leftarrow \gamma_\mu \mu_k$ and Θ_k by (11).
    Use Algorithm 2 to compute $(\overline{\alpha}_k, \overline{s}_k) \leftarrow$ CAUCHY_AL$(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
    Compute an approximate solution s_k to (10) satisfying (13a).
  end while
  Set $\alpha_k \leftarrow \gamma_\alpha^l$ where $l \geq 0$ is the smallest integer satisfying (14).
  Set $x_{k+1} \leftarrow x_k + \alpha_k s_k$.
  if $\|c_{k+1}\|_2 \leq t_j$, **then**
    Compute any $\widehat{y}_{k+1}$ satisfying (15).
    if $\min\{\|F_{\text{L}}(x_{k+1}, \widehat{y}_{k+1})\|_2, \|F_{\text{AL}}(x_{k+1}, y_k, \mu_k)\|_2\} \leq T_j$, **then**
      Set $k_j \leftarrow k+1$ and $Y_{j+1} \leftarrow \max\{Y, t_{j-1}^{-\epsilon}\}$.
      Set $t_{j+1} \leftarrow \min\{\gamma_t t_j, t_j^{1+\epsilon}\}$ and $T_{j+1} \leftarrow \gamma_T T_j$.
      Set y_{k+1} from (16) where α_y satisfies (17).
      Set $j \leftarrow j+1$.
    else
      Set $y_{k+1} \leftarrow y_k$.
    end if
  else
    Set $y_{k+1} \leftarrow y_k$.
  end if
  Set $\mu_{k+1} \leftarrow \mu_k$.
  Set $k \leftarrow k+1$.
end loop

are given in Appendices A and B. (The proofs of these results are similar to the corresponding results in [14]; for reference, complete details can be found in [15].) In order to show well-posedness of the algorithm, we make the following formal assumption.

Assumption 2.1 *At each given x_k, the objective function f and constraint function c are both twice-continuously differentiable.*

Under this assumption, we have the following theorem.

Theorem 2.2 *Suppose that Assumption 2.1 holds. Then the kth iteration of Algorithm 3 is well posed. That is, either the algorithm will terminate in line 8 or 11, or it will compute $\mu_k > 0$ such that $F_{AL}(x_k, y_k, \mu_k) \neq 0$ and for the steps $s_k = \bar{s}_k$ and $r_k = \bar{r}_k$ the conditions in (13) will be satisfied, in which case $(x_{k+1}, y_{k+1}, \mu_{k+1})$ will be computed.*

According to Theorem 2.2, we have that Algorithm 3 will either terminate finitely or produce an infinite sequence of iterates. If it terminates finitely—which can only occur if line 8 or 11 is executed—then the algorithm has computed a first-order stationary solution or an infeasible stationary point and there is nothing else to prove about the algorithm's performance in such cases. Therefore, it remains to focus on the global convergence properties of Algorithm 3 under the assumption that the sequence $\{(x_k, y_k, \mu_k)\}$ is infinite. For such cases, we make the following additional assumption.

Assumption 2.3 *The primal sequences $\{x_k\}$ and $\{x_k + s_k\}$ are contained in a convex compact set over which the objective function f and constraint function c are both twice-continuously differentiable.*

Our main global convergence result for Algorithm 3 is as follows.

Theorem 2.4 *Suppose that Assumptions 2.2 and 2.3 hold. Then one of the following must hold:*

(i) every limit point x_* of $\{x_k\}$ is an infeasible stationary point;

(ii) $\mu_k \nrightarrow 0$ and there exists an infinite ordered set $\mathcal{K} \subseteq \mathbb{N}$ such that every limit point of $\{(x_k, \widehat{y}_k)\}_{k \in \mathcal{K}}$ is first-order stationary for (1); or

(iii) $\mu_k \to 0$, every limit point of $\{x_k\}$ is feasible, and if there exists a positive integer p such that $\mu_{k_j - 1} \geq \gamma_\mu^p \mu_{k_{j-1} - 1}$ for all sufficiently large j, then there exists an infinite ordered set $\mathcal{J} \subseteq \mathbb{N}$ such that any limit point of either $\{(x_{k_j}, \widehat{y}_{k_j})\}_{j \in \mathcal{J}}$ or $\{(x_{k_j}, y_{k_j - 1})\}_{j \in \mathcal{J}}$ is first-order stationary for (1).

Observe that the conclusions are exactly the same as in [14, Theorem 3.14]. We also call the readers attention to the comments following [14, Theorem 3.14], which discuss the consequences of these results. In particular, these comments suggest how Algorithm 3 may be modified to guarantee convergence to first-order stationary points, even in case (iii) of the theorem. However, as mentioned in [14], we do not consider these modifications to the algorithm to have practical benefits. This perspective is supported by the numerical tests presented in the following section.

3 Numerical Experiments

In this section, we provide evidence that steering can have a positive effect on the performance of AL algorithms. To best illustrate the influence of steering, we implemented and tested algorithms in two pieces of software. First, in MATLAB, we implemented our adaptive AL line search algorithm, i.e., Algorithm 3, and the adaptive AL trust region method given as [14, Algorithm 4]. Since these methods were implemented from scratch, we had control over every aspect of the code, which allowed us to implement all features described in this paper and in [14]. Second, we implemented a simple modification of the AL trust region algorithm in the LANCELOT software package [12]. Our only modification to LANCELOT was to incorporate a basic form of steering; i.e., we did not change other aspects of LANCELOT, such as the mechanisms for triggering a multiplier update. In this manner, we were also able to isolate the effect that steering had on numerical performance, though it should be noted that there were differences between Algorithm 3 and our implemented algorithm in LANCELOT in terms of, e.g., the multiplier updates.

While we provide an extensive amount of information about the results of our experiments in this section, further information can be found in [15, Appendix C].

3.1 MATLAB implementation

3.1.1 Implementation details

Our MATLAB software was comprised of six algorithm variants. The algorithms were implemented as part of the same package so that most of the algorithmic components were exactly the same; the primary differences related to the step acceptance mechanisms and the manner in which the Lagrange multiplier estimates and penalty parameter were updated. First, for comparison against algorithms that utilized our steering mechanism, we implemented line search and trust region variants of a basic augmented Lagrangian method, given as [14, Algorithm 1]. We refer to these algorithms as `BAL-LS` (**b**asic **a**ugmented **L**agrangian, **l**ine **s**earch) and `BAL-TR` (**t**rust **r**egion), respectively. These algorithms clearly differed in that one used a line search and the other used a trust region strategy for step acceptance, but the other difference was that, like Algorithm 3 in this paper, `BAL-LS` employed a convexified model of the AL function. (We discuss more details about the use of this convexified model below.) The other algorithms implemented in our software included two variants of Algorithm 3 and two variants of [14, Algorithm 4]. The first variants of each, which we refer to as `AAL-LS` and `AAL-TR` (**a**daptive, as opposed to **b**asic), were straightforward implementations of these algorithms, whereas the latter variants, which we refer to as `AAL-LS-safe` and `AAL-TR-safe`, included an implementation of a safeguarding procedure for the steering mechanism. The safeguarding procedure will be described in detail shortly.

The main per-iteration computational expense for each algorithm variant can be attributed to the search direction computations. For computing a search direction via an approximate solve of (10) or [14, Prob. (3.8)], all algorithms essentially used the same procedure. For simplicity, all algorithms considered variants of these subproblems in which the ℓ_2-norm trust region was replaced by an ℓ_∞-norm trust region so that the subproblems were bound-constrained. (The same modification was used in the Cauchy step calculations.) Then, starting with the Cauchy step as the initial solution estimate and defining the initial working set by the bounds

identified as active by the Cauchy step, a projected conjugate gradient (PCG) method was used to compute an improved solution. During the PCG routine, if a new trial solution violated a bound constraint that was not already part of the working set, then this bound was added to the working set and the PCG routine was reinitialized. By contrast, if the reduced subproblem (corresponding to the current working set) was solved sufficiently accurately, then a check for termination was performed. In particular, multiplier estimates were computed for the working set elements. If these multiplier estimates were all nonnegative (or at least larger than a small negative number), then the subproblem was deemed to be solved and the routine terminated. Otherwise, an element corresponding to the most negative multiplier estimate was removed from the working set and the PCG routine was reinitialized. We do not claim that the precise manner in which we implemented this approach guaranteed convergence to an exact solution of the subproblem. However, the approach just described was based on well-established methods for solving bound-constrained quadratic optimization problems (QPs), and we found that it worked very well in our experiments. It should be noted that if, at any time, negative curvature was encountered in the PCG routine, then the solver terminated with the current PCG iterate. In this manner, the solutions were generally less accurate when negative curvature was encountered, but we claim that this did not have too adverse an effect on the performance of any of the algorithms.

A few additional comments are necessary to describe our search direction computation procedures. First, it should be noted that for the line search algorithms, the Cauchy step calculation in Algorithm 2 was performed with (12) as stated (i.e., with $\tilde{q}$), but the above PCG routine to compute the search direction was applied to (10) *without* the convexification for the quadratic term. However, we claim that this choice remains consistent with the stated algorithms since, for all algorithm variants, we performed a sanity check after the computation of the search direction. In particular, the reduction in the model of the AL function yielded by the search direction was compared against that yielded by the corresponding Cauchy step. If the Cauchy step actually provided a better reduction in the model, then the computed search direction was replaced by the Cauchy step. In this sanity check for the line search algorithms, we computed the model reductions *with* the convexification of the quadratic term (i.e., with $\tilde{q}$), which implies that, overall, our implemented algorithm guaranteed Cauchy decrease in the appropriate model for all algorithms. Second, we remark that for the algorithms that employed a steering mechanism, we did not employ the same procedure to approximately solve (7) or [14, Prob. (3.4)]. Instead, we simply used the Cauchy steps as approximate solutions of these subproblems. Finally, we note that in the steering mechanism, we checked condition (13c) with the Cauchy steps for each subproblem, despite the fact that the search direction was computed as a more accurate solution of (10) or [14, Prob. (3.8)]. This had the effect that the algorithms were able to modify the penalty parameter via the steering mechanism prior to computing the search direction; only Cauchy steps for the subproblems were needed for steering.

Most of the other algorithmic components were implemented similarly to the algorithm in [14]. As an example, for the computation of the estimates $\{\widehat{y}_{k+1}\}$ (which are required to satisfy (15)), we checked whether $\|F_{\mathrm{L}}(x_{k+1}, \pi(x_{k+1}, y_k, \mu_k))\|_2 \leq \|F_{\mathrm{L}}(x_{k+1}, y_k)\|_2$; if so, then we set $\widehat{y}_{k+1} \leftarrow \pi(x_{k+1}, y_k, \mu_k)$, and otherwise we set $\widehat{y}_{k+1} \leftarrow y_k$. Furthermore, for prescribed tolerances $\{\kappa_{\mathrm{opt}}, \kappa_{\mathrm{feas}}, \mu_{\min}\} \subset (0, \infty)$, we terminated an algorithm with a declaration that a

stationary point was found if

$$\|F_{\text{L}}(x_k, y_k)\|_\infty \leq \kappa_{\text{opt}} \quad \text{and} \quad \|c_k\|_\infty \leq \kappa_{\text{feas}}, \tag{18}$$

and terminated with a declaration that an infeasible stationary point was found if

$$\|F_{\text{FEAS}}(x_k)\|_\infty \leq \kappa_{\text{opt}}, \quad \|c_k\|_\infty > \kappa_{\text{feas}}, \quad \text{and} \quad \mu_k \leq \mu_{\min}. \tag{19}$$

As in [14], this latter set of conditions shows that we did not declare that an infeasible stationary point was found unless the penalty parameter had already been reduced below a prescribed tolerance. This helps in avoiding premature termination when the algorithm could otherwise continue and potentially find a point satisfying (18), which was always the preferred outcome. Each algorithm terminated with a message of failure if neither (18) nor (19) was satisfied within $k_{\max}$ iterations. It should also be noted that the problems were pre-scaled so that the ℓ_∞-norms of the gradients of the problem functions at the initial point would be less than or equal to a prescribed constant $G > 0$. The values for all of these parameters, as well as other input parameter required in the code, are summarized in Table 1. (Values for parameters related to updating the trust region radii required by [14, Algorithm 4] were set as in [14].)

Table 1: Input parameter values used in our MATLAB software.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
γ	0.5	κ_1	1	η_s	10^{-4}	κ_{feas}	10^{-5}
γ_μ	0.1	κ_2	1	η_{vs}	0.9	$\mu_{\min}$	10^{-8}
γ_α	0.5	κ_3	10^{-4}	ϵ	0.5	$k_{\max}$	10^4
γ_t	0.1	ε_r	10^{-4}	μ_0	1	G	10^2
γ_T	0.1	κ_t	0.9	κ_{opt}	10^{-5}		

We close this subsection with a discussion of some additional differences between the algorithms as stated in this paper and in [14] and those implemented in our software. We claim that none of these differences represents a significant departure from the stated algorithms; we merely made some adjustments to simplify the implementation and to incorporate features that we found to work well in our experiments. First, while all algorithms use the input parameter γ_μ given in Table 1 for decreasing the penalty parameter, we decrease the penalty parameter less significantly in the steering mechanism. In particular, in line 22 of Algorithm 3 and line 20 of [14, Algorithm 4], we replace γ_μ with 0.7. Second, in the line search algorithms, rather than set the trust region radii as in (8) and (11) where δ appears as a constant value, we defined a dynamic sequence, call it $\{\delta_k\}$, that depended on the step-size sequence $\{\alpha_k\}$. In this manner, δ_k replaced δ in (8) and (11) for all k. We initialized $\delta_0 \leftarrow 1$. Then, for all k, if $\alpha_k = 1$, then we set $\delta_{k+1} \leftarrow \frac{5}{3}\delta_k$, and if $\alpha_k < 1$, then we set $\delta_{k+1} \leftarrow \frac{1}{2}\delta_k$. Third, to simplify our implementation, we effectively ignored the imposed bounds on the multiplier estimates by setting $Y \leftarrow \infty$ and $Y_1 \leftarrow \infty$. This choice implies that we always chose $\alpha_y \leftarrow 1$ in (16). Fourth, we initialized the target values as

$$t_0 \leftarrow t_1 \leftarrow \max\{10^2, \min\{10^4, \|c_k\|_\infty\}\} \tag{20}$$

$$\text{and} \quad T_1 \leftarrow \max\{10^0, \min\{10^2, \|F_{\text{L}}(x_k, y_k)\|_\infty\}\}. \tag{21}$$

Finally, in `AAL-LS-safe` and `AAL-TR-safe`, we safeguard the steering procedure by shutting it off whenever the penalty parameter was smaller than a prescribed tolerance. Specifically, we considered the **while** condition in line 21 of Algorithm 3 and line 19 of [14, Algorithm 4] to be satisfied whenever $\mu_k \leq 10^{-4}$.

3.1.2 Results on CUTEst test problems

We tested our MATLAB algorithms on the subset of problems from the CUTEst [24] collection that have at least one general constraint and at most 1000 variables and 1000 constraints. This set contains 383 test problems. However, the results that we present in this section are only for those problems for which at least one of our six solvers obtained a successful result, i.e., where (18) or (19) was satisfied. This led to a set of 323 problems that are represented in the numerical results in this section.

To illustrate the performance of our MATLAB software, we use performance profiles as introduced by Dolan and Moré [17] to provide a visual comparison of different measures of performance. Consider a performance profile that measures performance in terms of required iterations until termination. For such a profile, if the graph associated with an algorithm passes through the point $(\alpha, 0.\beta)$, then, on $\beta\%$ of the problems, the number of iterations required by the algorithm was less than 2^α times the number of iterations required by the algorithm that required the fewest number of iterations. At the extremes of the graph, an algorithm with a higher value on the vertical axis may be considered a more efficient algorithm, whereas an algorithm on top at the far right of the graph may be considered more reliable. Since, for most problems, comparing values in the performance profiles for large values of α is not enlightening, we truncated the horizontal axis at 16 and simply remark on the numbers of failures for each algorithm.

Figures 1 and 2 show the results for the three line search variants, namely `BAL-LS`, `AAL-LS`, and `AAL-LS-safe`. The numbers of failures for these algorithms were 25, 3, and 16, respectively. The same conclusion may be drawn from both profiles: the steering variants (with and without safeguarding) were both more efficient and more reliable than the basic algorithm, where efficiency is measured by either the number of iterations (Figure 1) or the number of function evaluations (Figure 2) required. We display the profile for the number of function evaluations required since, for a line search algorithm, this value is always at least as large as the number of iterations, and will be strictly greater whenever backtracking is required to satisfy (14) (yielding $\alpha_k < 1$). From these profiles, one may observe that unrestricted steering (in `AAL-LS`) yielded superior performance to restricted steering (in `AAL-LS-safe`) in terms of both efficiency and reliability; this suggests that safeguarding the steering mechanism may diminish its potential benefits.

Figures 3 and 4 show the results for the three trust region variants, namely `BAL-TR`, `AAL-TR`, and `AAL-TR-safe`, the numbers of failures for which were 30, 12, and 20, respectively. Again, as for the line search algorithms, the same conclusion may be drawn from both profiles: the steering variants (with and without safeguarding) are both more efficient and more reliable than the basic algorithm, where now we measure efficiency by either the number of iterations (Figure 3) or the number of gradient evaluations (Figure 4) required before termination. We observe the number of gradient evaluations here (as opposed to the number of function evaluations) since, for a trust region algorithm, this value is never larger than the number of iterations, and will

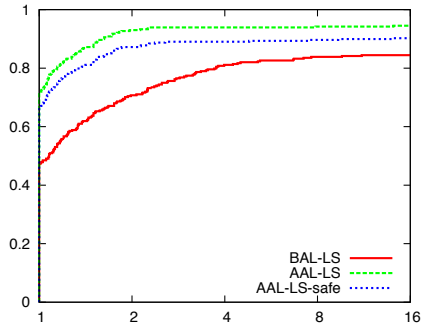

Figure 1: Performance profile for iterations: line search algorithms on the CUTEst set.

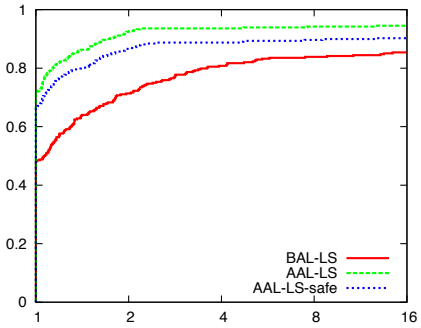

Figure 2: Performance profile for function evaluations: line search algorithms on the CUTEst set.

be strictly smaller whenever a step is rejected and the trust-region radius is decreased because of insufficient decrease in the AL function. These profiles also support the other observation that was made by the results for our line search algorithms, i.e., that unrestricted steering may be superior to restricted steering in terms of efficiency and reliability.

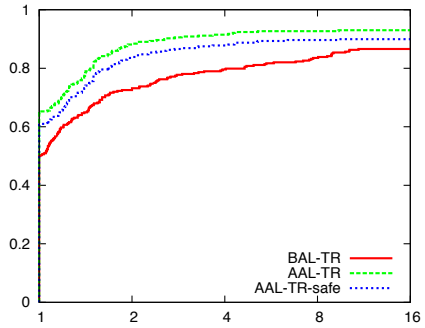

Figure 3: Performance profile for iterations: trust region algorithms on the CUTEst set.

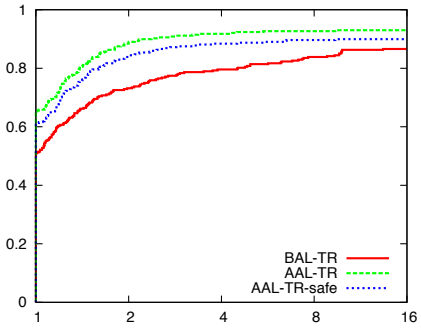

Figure 4: Performance profile for gradient evaluations: trust region algorithms on the CUTEst set.

The performance profiles in Figures 1–4 suggest that steering has practical benefits, and that safeguarding the procedure may limit its potential benefits. However, to be more confident in these claims, one should observe the final penalty parameter values typically produced by the algorithms. These observations are important since one may be concerned whether the algorithms that employ steering yield final penalty parameter values that are often significantly smaller than those yielded by basic AL algorithms. To investigate this possibility in our experiments, we collected the final penalty parameter values produced by all six algorithms; the results are in Table 2. The column titled μ_{final} gives a range for the final value of the penalty parameter. (For example, the value 27 in the `BAL-LS` column indicates that the final penalty parameter value computed by our basic line search AL algorithm fell in the range $[10^{-2}, 10^{-1})$ for 27 of the problems.)

We remark on two observations about the data in Table 2. First, as may be expected, the algorithms that employ steering typically reduce the penalty parameter below its initial value on some problems on which the other algorithms do not reduce it at all. This, in itself, is not a

Table 2: Numbers of CUTEst problems for which the final penalty parameter values were in the given ranges.

μ_{final}	BAL-LS	AAL-LS	AAL-LS-safe	BAL-TR	AAL-TR	AAL-TR-safe
1	139	87	87	156	90	90
$[10^{-1}, 1)$	43	33	33	35	46	46
$[10^{-2}, 10^{-1})$	27	37	37	28	29	29
$[10^{-3}, 10^{-2})$	17	42	42	19	49	49
$[10^{-4}, 10^{-3})$	22	36	36	18	29	29
$[10^{-5}, 10^{-4})$	19	28	42	19	25	39
$[10^{-6}, 10^{-5})$	15	19	11	9	11	9
$(0, 10^{-6})$	46	46	40	44	49	37

major concern, since a reasonable reduction in the penalty parameter may cause an algorithm to locate a stationary point more quickly. Second, we remark that the number of problems for which the final penalty parameter was very small (say, less than 10^{-4}) was similar for all algorithms, even those that employed steering. This suggests that while steering was able to aid in guiding the algorithms toward constraint satisfaction, the algorithms did not reduce the value to such a small value that feasibility became the only priority. Overall, our conclusion from Table 2 is that steering typically decreases the penalty parameter more than does a traditonal updating scheme, but one should not expect that the final penalty parameter value will be reduced unnecessarily small due to steering; rather, steering can have the intended benefit of improving efficiency and reliability by guiding a method toward constraint satisfaction more quickly.

3.1.3 Results on COPS test problems

We also tested our MATLAB software on the large-scale constrained problems available in the COPS [5] collection. This test set was designed to provide difficult test cases for nonlinear optimization software; the problems include examples from fluid dynamics, population dynamics, optimal design, mesh smoothing, and optimal control. For our purposes, we solved the smallest versions of the AMPL models [1, 19] provided in the collection. All of our solvers failed to solve the problems named *chain*, *dirichlet*, *henon*, *lane_emden*, and *robot1*, so these problems were excluded. The remaining set consisted of the following 17 problems: *bearing*, *camshape*, *catmix*, *channel*, *elec*, *gasoil*, *glider*, *marine*, *methanol*, *minsurf*, *pinene*, *polygon*, *rocket*, *steering*, *tetra*, *torsion*, and *triangle*. Since the size of this test set is relatively small, we have decided to display pair-wise comparisons of algorithms in the manner suggested in [30]. That is, for a performance measure of interest (e.g., number of iterations required until termination), we compare solvers, call them A and B, on problem j with the logarithmic *outperforming factor*

$$r^j_{AB} := -\log_2(\mathrm{m}^j_A/\mathrm{m}^j_B), \quad \text{where} \quad \begin{cases} \mathrm{m}^j_A & \text{is the measure for } A \text{ on problem } j \\ \mathrm{m}^j_B & \text{is the measure for } B \text{ on problem } j. \end{cases} \tag{22}$$

Therefore, if the measure of interest is iterations required, then $r^j_{AB} = p$ would indicate that solver A required 2^{-p} the iterations required by solver B. For all plots, we focus our attention

on the range $p \in [-2, 2]$.

The results of our experiments are given in Figures 5–8. For the same reasons as discussed in §3.1.2, we display results for iterations and function evaluations for the line search algorithms, and display results for iterations and gradient evaluations for the trust region algorithms. In addition, here we ignore the results for `AAL-LS-safe` and `AAL-TR-safe` since, as in the results in §3.1.2, we did not see benefits in safeguarding the steering mechanism. In each figure, a positive (negative) bar indicates that the algorithm whose name appears above (below) the horizontal axis yielded a better value for the measure on a particular problem. The results are displayed according to the order of the problems listed in the previous paragraph. In Figures 5 and 6 for the line search algorithms, the red bars for problems *catmix* and *polygon* indicate that `AAL-LS` failed on the former and `BAL-LS` failed on the latter; similarly, in Figures 7 and 8 for the trust region algorithms, the red bar for *catmix* indicates that `AAL-TR` failed on it.

The results in Figures 5 and 6 indicate that `AAL-LS` more often outperforms `BAL-LS` in terms of iterations and functions evaluations, though the advantage is not overwhelming. On the other hand, it is clear from Figures 7 and 8 that, despite the one failure, `AAL-TR` is generally superior to `BAL-TR`. We conclude from these results that steering was beneficial on this test set, especially in terms of the trust region methods.

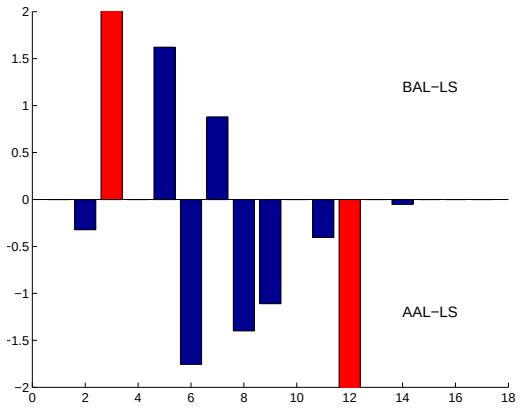

Figure 5: Outperforming factors for iterations: line search algorithms on the COPS set.

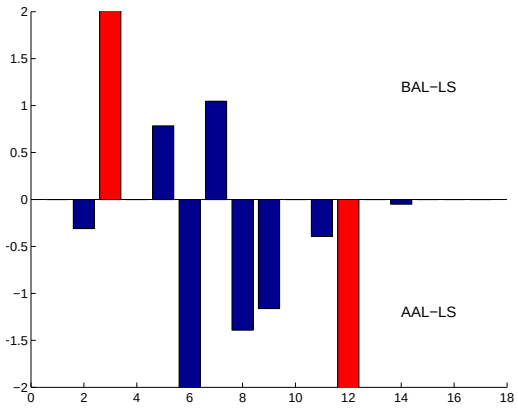

Figure 6: Outperforming factors for function evaluations: line search algorithms on the COPS set.

3.1.4 Results on optimal power flow (OPF) test problems

As a third and final set of experiments for our MATLAB software, we tested our algorithms on a collection of optimal power flow (OPF) problems modeled in AMPL using data sets obtained from MATPOWER [36]. OPF problems represent a challenging set of nonconvex problems. The active and reactive power flow and the network balance equations give rise to equality constraints involving nonconvex functions while the inequality constraints are linear and result from placing operating limits on quantities such as flows, voltages, and various control variables. The control variables include the voltages at generator buses and the active-power output of the generating units. The state variables consist of the voltage magnitudes and angles at each node as well as reactive and active flows in each link. Our test set was

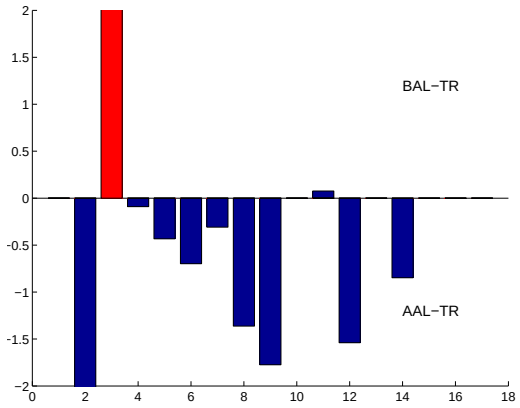

Figure 7: Outperforming factors for iterations: trust region algorithms on the COPS set.

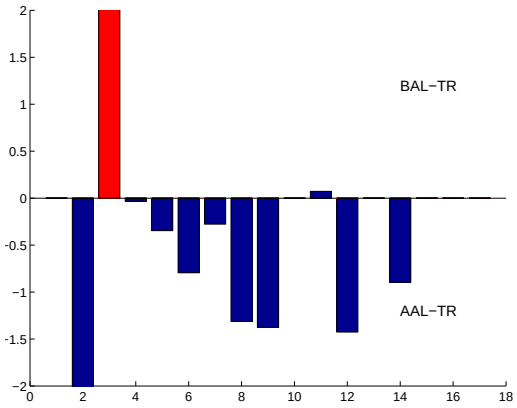

Figure 8: Outperforming factors for gradient evaluations: trust region algorithms on the COPS set.

comprised of 28 problems modeled on systems having 14 to 662 nodes from the IEEE test set. In particular, there are seven IEEE systems, each modeled in four different ways: (i) in Cartesian coordinates; (ii) in polar coordinates; (iii) with basic approximations to the sin and cos functions in the problem functions; and (iv) with linearized constraints based on DC power flow equations (in place of AC power flow). It should be noted that while linearizing the constraints in formulation (iv) led to a set of linear optimization problems, we still find it interesting to investigate the possible effect that steering may have in this context. All of the test problems were solved by all of our algorithm variants.

We provide outperforming factors in the same manner as in §3.1.3. Figures 9 and 10 reveal that `AAL-LS` typically outperforms `BAL-LS` in terms of both iterations and function evaluations, and Figures 11 and 12 reveal that `AAL-TR` more often than not outperforms `BAL-TR` in terms of iterations and gradient evaluations. Interestingly, these results suggest more benefits for steering in the line search algorithm than in the trust region algorithm, which is the opposite of that suggested by the results in §3.1.3. However, in any case, we believe that we have presented convincing numerical evidence that steering often has an overall beneficial effect on the performance of our MATLAB solvers.

3.2 An implementation of LANCELOT that uses steering

3.2.1 Implementation details

The results for our MATLAB software in the previous section illustrate that our adaptive line search AL algorithm and the adaptive trust region AL algorithm from [14] are often more efficient and reliable than basic AL algorithms that employ traditional penalty parameter and Lagrange multiplier updates. Recall, however, that our adaptive methods are different from their basic counterparts in two key ways. First, the steering conditions (13) are used to dynamically decrease the penalty parameter during the optimization process for the AL function. Second, our mechanisms for updating the Lagrange multiplier estimate are different than the basic algorithm outlined in [14, Algorithm 1] since they use optimality measures for both the Lagrangian and the AL functions (see line 30 of Algorithm 3) rather than only that

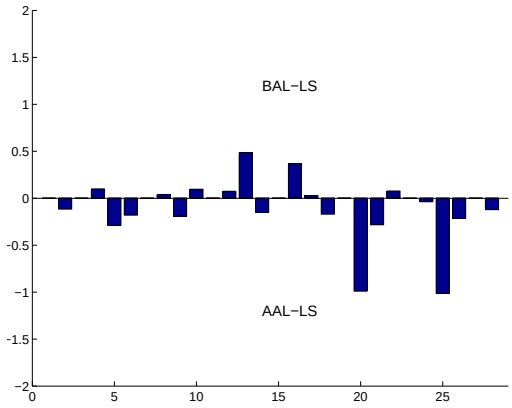

Figure 9: Outperforming factors for iterations: line search algorithms on OPF tests.

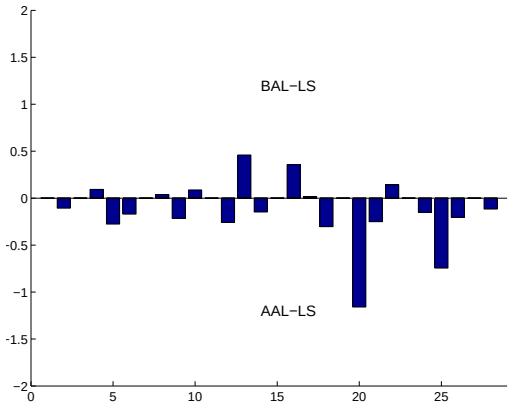

Figure 10: Outperforming factors for function evaluations: line search algorithms on OPF tests.

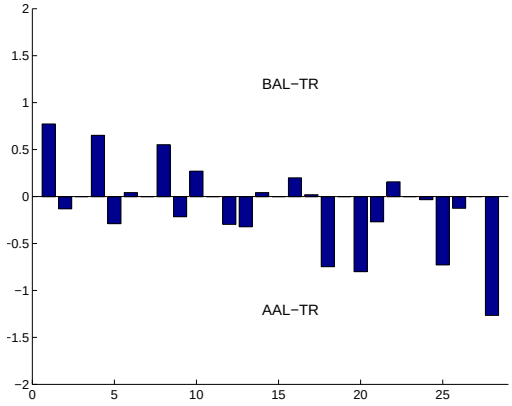

Figure 11: Outperforming factors for iterations: trust region algorithms on OPF tests.

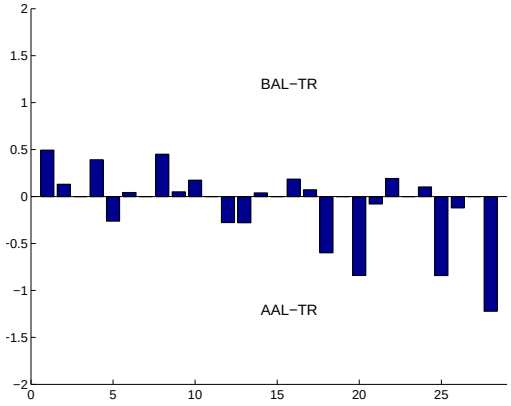

Figure 12: Outperforming factors for gradient evaluations: trust region algorithms on OPF tests.

for the AL function. We believe this strategy is more adaptive since it allows for updates to the Lagrange multipliers when the primal estimate is still far from a first-order stationary point for the AL function subject to the bounds.

In this section, we isolate the effect of the first of these differences by incorporating a steering strategy in the LANCELOT [12, 13] package that is available in the GALAHAD library [23]. Specifically, we made three principle enhancements in LANCELOT. First, along the lines of the model q in [14] and the convexified model $\tilde{q}$ defined in this paper, we defined the model $\hat{q} : \mathbb{R}^n \to \mathbb{R}$ of the AL function given by

$$\hat{q}(s; x, y, \mu) = s^T \nabla_x \ell\big(x, y + c(x)/\mu\big) + \tfrac{1}{2} s^T \big(\nabla_{xx} \ell(x, y) + J(x)^T J(x)/\mu\big) s$$

as an alternative to the Newton model $q_{\text{N}} : \mathbb{R}^n \to \mathbb{R}$, originally used in LANCELOT,

$$q_{\text{N}}(s; x, y, \mu) = s^T \nabla_x \ell(x, y + c(x)/\mu) + \tfrac{1}{2} s^T (\nabla_{xx} \ell(x, y + c(x)/\mu) + J(x)^T J(x)/\mu) s.$$

As in our adaptive algorithms, the purpose of employing such a model was to ensure that $\hat{q} \to q_v$ (pointwise) as $\mu \to 0$, which was required to ensure that our steering procedure was well-defined; see (A.1a). Second, we added routines to compute generalized Cauchy points [9] for both the constraint violation measure model q_v and $\hat{q}$ during the loop in which μ was decreased until the steering test (13c) was satisfied; recall the **while** loop starting on line 21 of Algorithm 3. Third, we used the value for μ determined in the steering procedure to compute a generalized Cauchy point for the Newton model $q_{\rm N}$, which was the model employed to compute the search direction. For each of the models just discussed, the generalized Cauchy point was computed using either an efficient sequential search along the piece-wise Cauchy arc [10] or via a backtracking Armijo search along the same arc [31]. We remark that this third enhancement would not have been needed if the model $\hat{q}$ were used to compute the search directions. However, in our experiments, it was revealed that using the Newton model typically led to better performance, so the results in this section were obtained using this third enhancement. In our implementation, the user was allowed to control which model was used via control parameters. We also added control parameters that allowed the user to restrict the number of times that the penalty parameter may be reduced in the steering procedure in a given iteration, and that disabled steering once the penalty parameter was reduced below a given tolerance (as in the safeguarding procedure implemented in our MATLAB software).

The new package was tested with three different control parameter settings. We refer to algorithm with the first setting, which did not allow any steering to occur, simply as `lancelot`. The second setting allowed steering to be used initially, but turned it off whenever $\mu \leq 10^{-4}$ (as in our safeguarded MATLAB algorithms). We refer to this variant as `lancelot-steering-safe`. The third setting allowed for steering to be used without any safeguards or restrictions; we refer to this variant as `lancelot-steering`. As in our MATLAB software, the penalty parameter was decreased by a factor of 0.7 until the steering test (13c) was satisfied. All other control parameters were set to their default `lancelot` values. The new package will be re-branded as LANCELOT in the next official release, GALAHAD 2.6.

GALAHAD was compiled with gfortran-4.7 with optimization -O and using Intel MKL BLAS. The code was executed on a single core of an Intel Xeon E5620 (2.4GHz) CPU with 23.5 GiB of RAM.

3.2.2 Results on CUTEST test problems

We tested `lancelot`, `lancelot-steering`, and `lancelot-steering-safe` on the subset of CUTEST problems that have at least one general constraint and at most 10,000 variables and 10,000 constraints. This amounted to 457 test problems. The results are displayed as performance profiles in Figures 13 and 14, which were created from the 364 of these problems that were solved by at least one of the algorithms. As in the previous sections, since the algorithms are trust region methods, we use the number of iterations and gradient evaluations required as the performance measures of interest.

We can make two important observations from these profiles. First, it is clear that `lancelot-steering` and `lancelot-steering-safe` yielded similar performance in terms of iterations and gradient evaluations, which suggests that safeguarding the steering mechanism is not necessary in practice. Second, `lancelot-steering` and `lancelot-steering-safe` were both more efficient and reliable than `lancelot` on these tests, thus showing the positive influ-

ence that steering can have on performance.

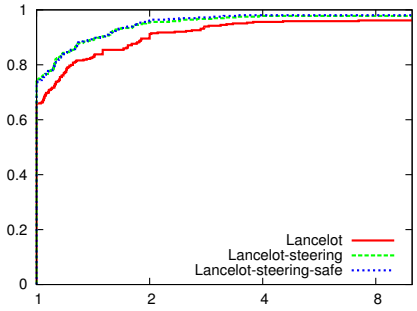

Figure 13: Performance profile for iterations: LANCELOT algorithms on the CUTEST set.

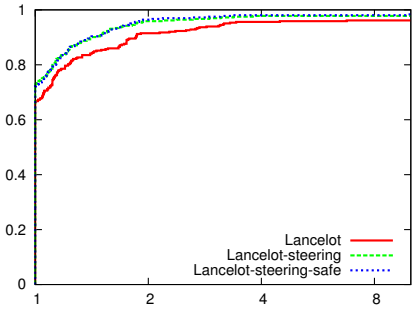

Figure 14: Performance profile for gradient evaluations: LANCELOT algorithms on the CUTEST set.

As in §3.1.2, it is important to observe the final penalty parameter values yielded by `lancelot-steering` and `lancelot-steering-safe` as opposed to those yielded by `lancelot`. For these experiments, we collected this information; see Table 3.

Table 3: Numbers of CUTEST problems for which the final penalty parameter values were in the given ranges.

μ_{final}	`lancelot`	`lancelot-steering`	`lancelot-steering-safe`
1	14	1	1
$[10^{-1}, 1)$	77	1	1
$[10^{-2}, 10^{-1})$	47	93	93
$[10^{-3}, 10^{-2})$	27	45	45
$[10^{-4}, 10^{-3})$	18	28	28
$[10^{-5}, 10^{-4})$	15	22	22
$[10^{-6}, 10^{-5})$	12	21	14
$(0, 10^{-6})$	19	18	25

We make a few remarks about the results in Table 3. First, as may have been expected, the `lancelot-steering` and `lancelot-steering-safe` algorithms typically reduced the penalty parameter below its initial value, even when `lancelot` did not reduce it at all throughout an entire run. Second, the number of problems for which the final penalty parameter was less than 10^{-4} was 171 for `lancelot` and 168 for `lancelot-steering`. Combining this fact with the previous observation leads us to conclude that steering tended to reduce the penalty parameter from its initial value of 1, but, overall, it did not decrease it much more aggressively than `lancelot`. Third, it is interesting to compare the final penalty parameter values for `lancelot-steering` and `lancelot-steering-safe`. Of course, these values were equal in any run in which the final penalty parameter was greater than or equal to 10^{-4}, since this was the threshold value below which safeguarding was activated. Interestingly, however, `lancelot-steering-safe` actually produced *smaller* values of the penalty parameter compared to `lancelot-steering` when the final penalty parameter was smaller than 10^{-4}. We initially found this observation to be somewhat counterintuitive, but we believe that it can be

explained by observing the penalty parameter updating strategy used by `lancelot`. (Recall that once safeguarding was activated in `lancelot-steering-safe`, the updating strategy became the same used in `lancelot`.) In particular, the decrease factor for the penalty parameter used in `lancelot` is 0.1, whereas the decrease factor used in steering the penalty parameter was 0.7. Thus, we believe that `lancelot-steering` reduced the penalty parameter more gradually once it was reduced below 10^{-4} while `lancelot-steering-safe` could only reduce it in the typical aggressive manner. (We remark that to (potentially) circumvent this inefficiency in `lancelot`, one could implement a different strategy in which the penalty parameter decrease factor is increased as the penalty parameter decreases, but in a manner that still ensures that the penalty parameter converges to zero when infinitely many decreases occur.) Overall, our conclusion from Table 3 is that steering typically decreases the penalty parameter more than a traditional updating scheme, but the difference is relatively small and we have implemented steering in a way that improves the overall efficiency and reliability of the method.

4 Conclusion

In this paper, we explored the numerical performance of adaptive updates to the Lagrange multiplier vector and penalty parameter in AL methods. Specific to the penalty parameter updating scheme is the use of steering conditions that guide the iterates toward the feasible region and toward dual feasibility in a balanced manner. Similar conditions were first introduced in [8] for exact penalty functions, but have been adapted in [14] and this paper to be appropriate for AL-based methods. Specifically, since AL methods are not exact (in that, in general, the trial steps do not satisfy linearized feasibility for any positive value of the penalty parameter), we allowed for a relaxation of the linearized constraints. This relaxation was based on obtaining a target level of infeasibility that is driven to zero at a modest, but acceptable, rate. This approach is in the spirit of AL algorithms since feasibility and linearized feasibility are only obtained in the limit. It should be noted that, like other AL algorithms, our adaptive methods can be implemented matrix-free, i.e., they only require matrix-vector products. This is of particular importance when solving large problems that have sparse derivative matrices.

As with steering strategies designed for exact penalty functions, our steering conditions proved to yield more efficient and reliable algorithms than a traditional updating strategy. This conclusion was made by performing a variety of numerical tests that involved our own MATLAB implementations and a simple modification of the well-known AL software LANCELOT. To test the potential for the penalty parameter to be reduced too quickly, we also implemented safeguarded variants of our steering algorithms. Across the board, our results indicate that safeguarding was not necessary and would typically degrade performance when compared to the unrestricted steering approach. We feel confident that these tests clearly show that although our theoretical global convergence guarantee is weaker than some algorithms (i.e., we cannot prove that the penalty parameter will remain bounded under a suitable constraint qualification), this should not be a concern in practice. Finally, we suspect that the steering strategies described in this paper would also likely improve the performance of other AL-based methods such as [4, 27].

Acknowledgments. We would like to thank Sven Leyffer and Victor Zavala from Argonne National Laboratory for providing us with the AMPL [1, 19] files required to test the optimal

power flow problems described in §3.1.4.

References

[1] AMPL Home Page, http://www.ampl.com.

[2] R. Andreani, E.G. Birgin, J.M. Martínez, and M.L. Schuverdt, *Augmented Lagrangian methods under the constant positive linear dependence constraint qualification*, Mathematical Programming 111 (2008), pp. 5–32, Available at http://dx.doi.org/10.1007/s10107-006-0077-1.

[3] E.G. Birgin and J.M. Martínez, *Augmented Lagrangian method with nonmonotone penalty parameters for constrained optimization*, Computational Optimization and Applications 51 (2012), pp. 941–965, Available at http://dx.doi.org/10.1007/s10589-011-9396-0.

[4] E.G. Birgin and J.M. Martínez, *Practical Augmented Lagrangian Methods for Constrained Optimization*, Fundamentals of Algorithms, SIAM, Philadelphia, PA, USA, 2014.

[5] A. Bondarenko, D. Bortz, and J.J. Moré, *COPS: Large-scale nonlinearly constrained optimization problems*, Technical Report ANL/MCS-TM-237, Mathematics and Computer Science division, Argonne National Laboratory, Argonne, IL, 1998, revised October 1999.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Foundations and Trends in Machine Learning 3 (2011), pp. 1–122.

[7] R.H. Byrd, G. Lopez-Calva, and J. Nocedal, *A line search exact penalty method using steering rules*, Mathematical Programming 133 (2012), pp. 39–73.

[8] R.H. Byrd, J. Nocedal, and R.A. Waltz, *Steering exact penalty methods for nonlinear programming*, Optimization Methods and Software 23 (2008), pp. 197–213, Available at http://dx.doi.org/10.1080/10556780701394169.

[9] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *Global convergence of a class of trust region algorithms for optimization with simple bounds*, SIAM Journal on Numerical Analysis 25 (1988), pp. 433–460.

[10] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Mathematics of Computation 50 (1988), pp. 399–430.

[11] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds*, SIAM Journal on Numerical Analysis 28 (1991), pp. 545–572.

[12] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, LANCELOT*: A Fortran package for large-scale nonlinear optimization (Release A)*, Lecture Notes in Computation Mathematics 17, Springer Verlag, Berlin, Heidelberg, New York, London, Paris and Tokyo, 1992.

[13] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *Numerical experiments with the* LANCELOT *package (Release A) for large-scale nonlinear optimization*, Mathematical Programming 73 (1996), pp. 73–110.

[14] F.E. Curtis, H. Jiang, and D.P. Robinson, *An adaptive augmented Lagrangian method for large-scale constrained optimization*, Mathematical Programming (2014), Available at http://dx.doi.org/10.1007/s10107-014-0784-y.

[15] F.E. Curtis, N.I.M. Gould, H. Jiang, and D.P. Robinson, *Adaptive augmented Lagrangian methods: Algorithms and practical numerical experience*, Available at http://xxx.tau.ac.il/abs/1408.4500, arXiv:1408.4500.

[16] K.R. Davidson and A.P. Donsig, *Real Analysis and Applications*, Undergraduate Texts in Mathematics, Springer, New York, 2010, Available at http://dx.doi.org/10.1007/978-0-387-98098-0.

[17] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming 91 (2002), pp. 201–213.

[18] D. Fernández and M.V. Solodov, *Local convergence of exact and inexact augmented Lagrangian methods under the second-order sufficiency condition*, SIAM Journal on Optimization 22 (2012), pp. 384–407.

[19] R. Fourer, D.M. Gay, and B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Brooks/Cole—Thomson Learning, Pacific Grove, USA, 2003.

[20] D. Gabay and B. Mercier, *A dual algorithm for the solution of nonlinear variational problems via finite element approximations*, Computers and Mathematics with Applications 2 (1976), pp. 17–40.

[21] R. Glowinski and A. Marroco, *Sur l'Approximation, par Elements Finis d'Ordre Un, el la Resolution, par Penalisation-Dualité, d'une Classe de Problèmes de Dirichlet Nonlineares*, Revue Française d'Automatique, Informatique et Recherche Opérationelle 9 (1975), pp. 41–76.

[22] 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 (2003), pp. 373–394.

[23] N.I.M. Gould, D. Orban, and Ph.L. Toint, GALAHAD*—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization*, ACM Transactions on Mathematical Software 29 (2003), pp. 353–372.

[24] N.I.M. Gould, D. Orban, and Ph.L. Toint, CUTEst*: A constrained and unconstrained testing environment with safe threads*, Tech. Rep. RAL-TR-2013-005, Rutherford Appleton Laboratory, 2013.

[25] M.R. Hestenes, *Multiplier and gradient methods*, Journal of Optimization Theory and Applications 4 (1969), pp. 303–320.

[26] A.F. Izmailov and M.V. Solodov, *On attraction of linearly constrained Lagrangian methods and of stabilized and quasi-Newton SQP methods to critical multipliers*, Mathematical Programming 126 (2011), pp. 231–257, Available at http://dx.doi.org/10.1007/s10107-009-0279-4.

[27] M. Kočvara and M. Stingl, *PENNON: A code for convex nonlinear and semidefinite programming*, Optimization Methods and Software 18 (2003), pp. 317–333.

[28] M. Kočvara and M. Stingl, *PENNON: A generalized augmented Lagrangian method for semidefinite programming*, in *High Performance Algorithms and Software for Nonlinear Optimization (Erice, 2001)*, Applied Optimization, Vol. 82, Kluwer Academic Publishing, Norwell, MA, 2003, pp. 303–321, Available at http://dx.doi.org/10.1007/978-1-4613-0241-4_14.

[29] M. Mongeau and A. Sartenaer, *Automatic decrease of the penalty parameter in exact penalty function methods*, European Journal of Operational Research 83 (1995), pp. 686–699.

[30] J.L. Morales, *A numerical study of limited memory BFGS methods*, Applied Mathematics Letters 15 (2002), pp. 481–487.

[31] J.J. Moré, *Trust regions and projected gradients*, in *System Modelling and Optimization*, Vol. 113, lecture Notes in Control and Information Sciences, Springer Verlag, Heidelberg, Berlin, New York, 1988, pp. 1–13.

[32] M.J.D. Powell, *A method for nonlinear constraints in minimization problems*, in *Optimization*, Academic Press, London and New York, 1969, pp. 283–298.

[33] Z. Qin, D. Goldfarb, and S. Ma, *An alternating direction method for total variation denoising*, arXiv preprint arXiv:1108.1587 (2011).

[34] Ph.L. Toint, *Nonlinear stepsize control, trust regions and regularizations for unconstrained optimization*, Optimization Methods and Software 28 (2013), pp. 82–95, Available at http://www.tandfonline.com/doi/abs/10.1080/10556788.2011.610458.

[35] J. Yang, Y. Zhang, and W. Yin, *A fast alternating direction method for TVL1-L2 signal reconstruction from partial Fourier data*, IEEE Journal of Selected Topics in Signal Processing 4 (2010), pp. 288–297.

[36] R.D. Zimmerman, C.E. Murillo-Sánchez, and R.J. Thomas, *Matpower: Steady-state operations, planning, and analysis tools for power systems research and education*, Power Systems, IEEE Transactions on 26 (2011), pp. 12–19.

Appendix A: Well-posedness

Our goal in this appendix is to prove that Algorithm 3 is well-posed under Assumption 2.1. Since this assumption is assumed to hold throughout the remainder of this appendix, we do not refer to it explicitly in the statement of each lemma and proof.

A.1 Preliminary results

Our proof of the well-posedness of Algorithm 3 relies on showing that it will either terminate finitely or will produce an infinite sequence of iterates $\{(x_k, y_k, \mu_k)\}$. In order to show this, we first require that the **while** loop that begins at line 13 of Algorithm 3 terminates finitely. Since the same loop appears in the AL trust region method in [14] and the proof of the result in the case of that algorithm is the same as that for Algorithm 3, we need only refer to the result in [14] in order to state the following lemma for Algorithm 3.

Lemma A.1 ([14, Lemma 3.2]) *If line 13 is reached, then $F_{AL}(x_k, y_k, \mu) \neq 0$ for all sufficiently small $\mu > 0$.*

Next, since the Cauchy steps employed in Algorithm 3 are similar to those employed in the method in [14], we may state the following lemma showing that Algorithms 1 and 2 are well defined when called in lines 17, 19, and 23 of Algorithm 3. It should be noted that a slight difference between Algorithm 2 and the similar procedure in [14] is the use of the convexified model $\tilde{q}$ in (12). However, we claim that this difference does not affect the veracity of the result.

Lemma A.2 ([14, Lemma 3.3]) *The following hold true:*

(i) The computation of $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k)$ in line 17 is well defined and yields $\Gamma_k \in (1, 2]$ and $\varepsilon_k \in [0, \varepsilon_r)$.

(ii) The computation of $(\overline{\alpha}_k, \overline{s}_k)$ in lines 19 and 23 is well defined.

The next result highlights critical relationships between q_v and $\tilde{q}$ as $\mu \to 0$.

Lemma A.3 ([15, Lemma A.3]) *Let $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) \leftarrow$ CAUCHY_FEASIBILITY(x_k, θ_k) with θ_k defined by (8) and, as quantities dependent on the penalty parameter $\mu > 0$, let $(\overline{\alpha}_k(\mu), \overline{s}_k(\mu)) \leftarrow$ CAUCHY_AL$(x_k, y_k, \mu, \Theta_k(\mu), \varepsilon_k)$ with $\Theta_k(\mu) := \Gamma_k \delta \|F_{AL}(x_k, y_k, \mu)\|_2$ (see (11)). Then, the following hold true:*

$$\lim_{\mu \to 0} \left(\max_{\|s\|_2 \leq 2\theta_k} |\tilde{q}(s; x_k, y_k, \mu) - q_v(s; x_k)| \right) = 0, \tag{A.1a}$$

$$\lim_{\mu \to 0} \nabla_x \mathcal{L}(x_k, y_k, \mu) = J_k^T c_k, \tag{A.1b}$$

$$\lim_{\mu \to 0} \overline{s}_k(\mu) = \overline{r}_k, \tag{A.1c}$$

$$\text{and} \quad \lim_{\mu \to 0} \Delta q_v(\overline{s}_k(\mu); x_k) = \Delta q_v(\overline{r}_k; x_k). \tag{A.1d}$$

We also need the following lemma related to Cauchy decreases in the models q_v and $\tilde{q}$.

Lemma A.4 ([15, Lemma A.4]) *Let Ω be any scalar value such that*

$$\Omega \geq \max\{\|\mu_k \nabla^2_{xx} \ell(x_k, y_k) + J_k^T J_k\|_2, \|J_k^T J_k\|_2\}. \tag{A.2}$$

Then, the following hold true:

(i) For some $\kappa_4 \in (0,1)$*, the Cauchy step for subproblem* (7) *yields*

$$\Delta q_v(\overline{r}_k; x_k) \geq \kappa_4 \|F_{FEAS}(x_k)\|_2^2 \min\left\{\delta, \frac{1}{1+\Omega}\right\}. \tag{A.3}$$

(ii) For some $\kappa_5 \in (0,1)$*, the Cauchy step for subproblem* (10) *yields*

$$\Delta \tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) \geq \kappa_5 \|F_{AL}(x_k, y_k, \mu_k)\|_2^2 \min\left\{\delta, \frac{1}{1+\Omega}\right\}. \tag{A.4}$$

The next lemma shows that the **while** loop at line 21, which is responsible for ensuring that our adaptive steering conditions in (13) are satisfied, terminates finitely.

Lemma A.5 ([15, Lemma A.5]) *The* ***while*** *loop that begins at line 21 of Algorithm 3 terminates finitely.*

The final lemma of this section shows that s_k is a strict descent direction for the AL function. The conclusion of this lemma is the primary motivation for our use of the convexified model $\tilde{q}$.

Lemma A.6 ([15, Lemma A.6]) *At line 26 of Algorithm 3, the search direction* s_k *is a strict descent direction for* $\mathcal{L}(\cdot, y_k, \mu_k)$ *from* x_k*. In particular,*

$$\nabla_x \mathcal{L}(x_k, y_k, \mu_k)^T s_k \leq -\Delta \tilde{q}(s_k; x_k, y_k, \mu_k) \leq -\kappa_1 \Delta \tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) < 0. \tag{A.5}$$

A.2 Proof of well-posedness result

Proof of Theorem 2.2. If, during the kth iteration, Algorithm 3 terminates in line 8 or 11, then there is nothing to prove. Thus, to proceed in the proof, we may assume that line 13 is reached. Lemma A.1 then ensures that

$$F_{\rm AL}(x_k, y_k, \mu) \neq 0 \ \text{ for all sufficiently small } \ \mu > 0. \tag{A.6}$$

Consequently, the **while** loop in line 13 will terminate for a sufficiently small $\mu_k > 0$. Next, by construction, conditions (13a) and (13b) are satisfied for any $\mu_k > 0$ by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. Lemma A.5 then shows that for a sufficiently small $\mu_k > 0$, (13c) is also satisfied by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. Therefore, line 26 will be reached. Finally, Lemma A.6 ensures that α_k in line 26 is well-defined. This completes the proof as all remaining lines in the kth iteration are explicit. □

Appendix B: Global Convergence

We shall tacitly presume that Assumption 2.3 holds throughout this section, and not state it explicitly. This assumption and the bound on the multipliers enforced in line 33 of Algorithm 3 imply that there exists a positive monotonically increasing sequence $\{\Omega_j\}_{j\geq 1}$ such that for all $k_j \leq k < k_{j+1}$ we have

$$\|\nabla^2_{xx}\mathcal{L}(\sigma, y_k, \mu_k)\|_2 \leq \Omega_j \ \text{ for all } \sigma \text{ on the segment } [x_k, x_k + s_k], \tag{B.1a}$$

$$\|\mu_k \nabla^2_{xx}\ell(x_k, y_k) + J_k^T J_k\|_2 \leq \Omega_j, \tag{B.1b}$$

$$\text{and } \ \|J_k^T J_k\|_2 \leq \Omega_j. \tag{B.1c}$$

In the subsequent analysis, we make use of the subset of iterations for which line 31 of Algorithm 3 is reached. For this purpose, we define the iteration index set

$$\mathcal{Y} := \{k_j : \|c_{k_j}\|_2 \leq t_j,\ \min\{\|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2, \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2\} \leq T_j\}. \tag{B.2}$$

B.1 Preliminary results

The following result provides critical bounds on differences in (components of) the augmented Lagrangian summed over sequences of iterations. We remark that the proof in [14] essentially relies on Assumption 2.3 and Dirichlet's Test [16, §3.4.10].

Lemma B.1 ([14, Lemma 3.7].) *The following hold true.*

(i) If $\mu_k = \mu$ for some $\mu > 0$ and all sufficiently large k, then there exist positive constants M_f, M_c, and $M_{\mathcal{L}}$ such that for all integers $p \geq 1$ we have

$$\sum_{k=0}^{p-1} \mu_k (f_k - f_{k+1}) < M_f, \tag{B.3}$$

$$\sum_{k=0}^{p-1} \mu_k y_k^T (c_{k+1} - c_k) < M_c, \tag{B.4}$$

$$\text{and} \quad \sum_{k=0}^{p-1} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) < M_{\mathcal{L}}. \tag{B.5}$$

(ii) If $\mu_k \to 0$, then the sums

$$\sum_{k=0}^{\infty} \mu_k (f_k - f_{k+1}), \tag{B.6}$$

$$\sum_{k=0}^{\infty} \mu_k y_k^T (c_{k+1} - c_k), \tag{B.7}$$

$$\text{and} \quad \sum_{k=0}^{\infty} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) \tag{B.8}$$

converge and are finite, and

$$\lim_{k \to \infty} \|c_k\|_2 = \bar{c} \ \text{ for some } \bar{c} \geq 0. \tag{B.9}$$

We also need the following lemma that bounds the step-size sequence $\{\alpha_k\}$ below.

Lemma B.2 *There exists a positive monotonically decreasing sequence $\{C_j\}_{j \geq 1}$ such that, with the sequence $\{k_j\}$ computed in Algorithm 3, the step-size sequence $\{\alpha_k\}$ satisfies*

$$\alpha_k \geq C_j > 0 \ \text{ for all } \ k_j \leq k < k_{j+1}.$$

Proof of Lemma B.2. By Taylor's Theorem and Lemma A.6, it follows under Assumption 2.3 that there exists $\tau > 0$ such that for all sufficiently small $\alpha > 0$ we have

$$\mathcal{L}(x_k + \alpha s_k, y_k, \mu_k) - \mathcal{L}(x_k, y_k, \mu_k) \leq -\alpha \Delta \tilde{q}(s_k; x_k, y_k, \mu_k) + \tau \alpha^2 \|s_k\|^2. \tag{B.10}$$

On the other hand, during the line search implicit in line 26 of Algorithm 3, a step-size α is rejected if

$$\mathcal{L}(x_k + \alpha s_k, y_k, \mu_k) - \mathcal{L}(x_k, y_k, \mu_k) > -\eta_s \alpha \Delta\tilde{q}(s_k; x_k, y_k, \mu_k). \tag{B.11}$$

Combining (B.10), (B.11), and (13a) we have that a rejected step-size α satisfies

$$\alpha > \frac{(1-\eta_s)\Delta\tilde{q}(s_k; x_k, y_k, \mu_k)}{\tau\|s_k\|_2^2} \geq \frac{(1-\eta_s)\Delta\tilde{q}(s_k; x_k, y_k, \mu_k)}{\tau\Theta_k^2}.$$

From this bound, the fact that if the line search rejects a step-size it multiplies it by $\gamma_\alpha \in (0,1)$, (13a), (A.4), (B.1b), (11), and $\Gamma_k \in (1,2]$ (see Lemma A.2) it follows that, for all $k \in [k_j, k_{j+1})$,

$$\begin{aligned}
\alpha_k &\geq \frac{\gamma_\alpha(1-\eta_s)\Delta\tilde{q}(s_k; x_k, y_k, \mu_k)}{\tau\Theta_k^2} \\
&\geq \frac{\gamma_\alpha(1-\eta_s)\kappa_1\kappa_5\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2^2}{\tau\Gamma_k^2\delta^2\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2^2} \min\left\{\delta, \frac{1}{1+\Omega_j}\right\} \\
&\geq \frac{\gamma_\alpha(1-\eta_s)\kappa_1\kappa_5}{4\tau\delta^2} \min\left\{\delta, \frac{1}{1+\Omega_j}\right\} =: C_j > 0,
\end{aligned}$$

as desired. □

We break the remainder of the analysis into two cases depending on whether there are a finite or an infinite number of modifications of the Lagrange multiplier estimate.

B.2 A finite number of multiplier updates

In this section, we suppose that the set $\mathcal{Y}$ in (B.2) is finite in that the counter j in Algorithm 3 satisfies

$$j \in \{1, 2, \ldots \bar{j}\} \text{ for some finite } \bar{j}. \tag{B.12}$$

This allows us to define, and consequently use in our analysis, the quantities

$$t := t_{\bar{j}} > 0 \text{ and } T := T_{\bar{j}} > 0. \tag{B.13}$$

We provide two lemmas in this subsection. The first considers cases when the penalty parameter converges to zero, and the second considers cases when the penalty parameter remains bounded away from zero. This first case—in which the multiplier estimate is only modified a finite number of times and the penalty parameter vanishes—may be expected to occur when (1) is infeasible. Indeed, in this case, we show that every limit point of the primal iterate sequence is an infeasible stationary point.

Lemma B.3 ([15, Lemma B.3]) *If $|\mathcal{Y}| < \infty$ and $\mu_k \to 0$, then there exist a vector y and integer $\bar{k} \geq 0$ such that*

$$y_k = y \text{ for all } k \geq \bar{k}, \tag{B.14}$$

and for some constant $\bar{c} > 0$, we have the limits

$$\lim_{k\to\infty} \|c_k\|_2 = \bar{c} > 0 \text{ and } \lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{B.15}$$

Therefore, every limit point of $\{x_k\}_{k\geq 0}$ is an infeasible stationary point.

The next lemma considers the case when μ stays bounded away from zero. This is possible, for example, if the algorithm converges to an infeasible stationary point that is stationary for the AL function for the final Lagrange multiplier estimate and penalty parameter computed in the algorithm.

Lemma B.4 ([15, Lemma B.4]) *If $|\mathcal{Y}| < \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then with t defined in (B.13) there exist a vector y and integer $\overline{k} \geq 0$ such that*

$$y_k = y \quad \text{and} \quad \|c_k\|_2 \geq t \quad \text{for all} \quad k \geq \overline{k}, \tag{B.16}$$

and we have the limit

$$\lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{B.17}$$

Therefore, every limit point of $\{x_k\}_{k\geq 0}$ is an infeasible stationary point.

This completes the analysis for the case that the set $\mathcal{Y}$ is finite.

B.3 An infinite number of multiplier updates

We now suppose that $|\mathcal{Y}| = \infty$. In this case, it follows from the procedures for updating the Lagrange multiplier estimate and target values in Algorithm 3 that

$$\lim_{j\to\infty} t_j = \lim_{j\to\infty} T_j = 0. \tag{B.18}$$

As in the previous subsection, we split the analysis in this subsection into two results. This time, we begin by considering the case when the penalty parameter remains bounded below and away from zero. In this scenario, we state the following result that a subsequence of the iterates converges to a first-order stationary point.

Lemma B.5 ([14, Lemma 3.10].) *If $|\mathcal{Y}| = \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then*

$$\lim_{j\to\infty} c_{k_j} = 0 \tag{B.19a}$$

$$\text{and} \quad \lim_{j\to\infty} F_L(x_{k_j}, \widehat{y}_{k_j}) = 0. \tag{B.19b}$$

Thus, any limit point (x_*, y_*) of $\{(x_{k_j}, \widehat{y}_{k_j})\}_{j\geq 0}$ is first-order stationary for (1).

Finally, we consider the case when the penalty parameter converges to zero.

Lemma B.6 ([14, Lemma 3.13]) *If $|\mathcal{Y}| = \infty$ and $\mu_k \to 0$, then*

$$\lim_{k\to\infty} c_k = 0. \tag{B.20}$$

If, in addition, there exists a positive integer p such that $\mu_{k_j-1} \geq \gamma_\mu^p \mu_{k_{j-1}-1}$ for all sufficiently large j, then there exists an infinite ordered set $\mathcal{J} \subseteq \mathbb{N}$ such that

$$\lim_{j\in\mathcal{J}, j\to\infty} \|F_L(x_{k_j}, \widehat{y}_{k_j})\|_2 = 0 \quad \text{or} \quad \lim_{j\in\mathcal{J}, j\to\infty} \|F_L(x_{k_j}, \pi(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}))\|_2 = 0. \tag{B.21}$$

In such cases, if the first (respectively, second) limit in (B.21) holds, then along with (B.20) it follows that any limit point of $\{(x_{k_j}, \widehat{y}_{k_j})\}_{j\in\mathcal{J}}$ (respectively, $\{(x_{k_j}, y_{k_j-1})\}_{j\in\mathcal{J}}$) is a first-order stationary point for (1).

B.4 Proof of global convergence result

Proof of Theorem 2.4. Lemmas B.3, B.4, B.5 and B.6 cover the only four possible outcomes of Algorithm 3; the result follows from those described in these lemmas. □