AUGMENTED LAGRANGIAN BASED ALGORITHMS FOR NONCONVEX OPTIMIZATION WITH APPLICATIONS IN SUBSPACE CLUSTERING

by

Hao Jiang

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

May, 2016

Abstract

This thesis considers the design, analysis, and implementation of algorithms for nonconvex optimization that utilize the augmented Lagrangian function.

In the first part of the thesis, we address general nonconvex optimization problems that have smooth objective and constraint functions. Observing a potential drawback of a traditional augmented Lagrangian (AL) method, we propose *adaptive* trust-region and linesearch AL algorithms that use the same novel feature, namely, an adaptive update for the penalty parameter. As with a traditional AL algorithm, the adaptive methods are matrix-free (i.e., they do not need to form or factorize problem matrices) and thus represent a viable option for solving large-scale problems. We prove global convergence for our adaptive AL algorithms and illustrate through extensive numerical experiments that our methods outperform traditional AL methods in terms of efficiency and reliability.

In the latter part of the thesis, we focus on a structured nonconvex nonsmooth problem arising from a machine learning application called subspace clustering. We show that the alternating direction method of multipliers (ADMM), which may

ABSTRACT

roughly be described as an application of block-wise coordinate minimization of the AL function, is well suited for this machine learning task. Moreover, we establish a global convergence result for the algorithm since it was previously unknown. Numerical results are presented to show that the chosen optimization modeling formulation together with ADMM can achieve subspace clustering accuracy that is on par with other state-of-the-art methods.

Primary Reader: Daniel P. Robinson

Secondary Reader: Amitabh Basu

Acknowledgments

I would like to first thank Daniel P. Robinson, my advisor, for his guidance and support throughout the years of my graduate study. He was always available for any kind of help and without his patience and confidence in me, this dissertation would have been impossible.

I am also lucky to have the opportunity to work with Frank E. Curtis from Lehigh University. Without his knowledge, expertise and guidance, the accomplishments in chapters 3, 4 and 5 would not have been possible.

I also wish to thank René Vidal, for sharing his knowledge and expertise in subspace clustering. His guidance and stimulating advices are among the main reasons of the accomplishments in Chapter 6.

I am grateful to Nick Gould from Rutherford Appleton Laboratory for his help in the numerical studies in Chapter 5; Sven Leyffer and Victor Zavala from Argonne National Laboratory for providing us with the AMPL [1, 2] files required to test the optimal power flow problems described in Section 5.1.4; and Amitabh Basu for serving on my committee and being the second reader of this dissertation.

ACKNOWLEDGMENTS

Finally, I am grateful to my parents for always being there for me during my graduate study. Their love and support gave me strength to complete this long journey.

Contents

List of Tables

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

Continuous optimization is ubiquitous in a variety of applications such as optimal control [3–5], resource allocation [6, 7], structural engineering [8, 9], machine learning [10,11], just to name a few. In many cases, the success of a task depends essentially on whether we can solve an optimization problem in a timely fashion. Recent years have witnessed many exciting advances in the field of numerical optimization. However, with modern optimization problems becoming more complex with increasingly larger scale, it is crucial for researchers to design new algorithms that can find reasonable solutions both efficiently and reliably. This dissertation concerns the design and analysis of iterative algorithms for nonlinear optimization problems that are nonconvex, constrained and of large scale.

A large part of the dissertation is devoted to two algorithms that are designed to

solve general constrained optimization problems of the form

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) \\ &\text{subject to} \;\; c_1(x) = 0,\; c_2(x) \geq 0, \end{aligned} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, and $c_1 : \mathbb{R}^n \to \mathbb{R}^{m_1}$ and $c_2 : \mathbb{R}^n \to \mathbb{R}^{m_2}$ are constraint functions. A traditional and popular class of methods for solving problem (1.1) is based on the augmented Lagrangian (AL) function. The popularity of such methods can, in part, be attributed to the fact that AL algorithms can be implemented matrix-free, meaning that it is not necessary to explicitly form or factorize a matrix. Moreover, it possesses reasonably fast local convergence guarantees under relatively weak assumptions.[12,13] However, we will see in Chapter 2 that an AL algorithm can potentially be inefficient, especially during early iterations, due to certain issues related to the traditional algorithmic design. One contribution of this dissertation is to address this issue. More specifically, we develop and analyze two new adaptive AL algorithms that overcome a key weakness of basic AL methods: they adjust their parameters in a slow and stagnant manner. We will show how these parameters may be updated in a much more efficient and adaptive manner, which is why we refer to our new algorithms as adaptive AL methods.

In many applications, the problems we want to solve may have special structure. In such cases, algorithms specifically tailored to the problem structure may be favored over all-purpose/black-box algorithms. One particular problem formulation

that frequently appears in engineering applications is the following:

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f_1(x) + f_2(z) \\ \text{subject to} \quad & Ax + Bz = d \end{aligned} \tag{1.2}$$

where the objective function is separable and there are only linear equality constraints. Besides the problems that are naturally formulated as (1.2), in many cases an optimization problem can be transformed into this form. For instance, in applications such as machine learning, people are often faced with the task of minimizing the summation of two functions, $f_1(x) + f_2(x)$, where function f_1 is some loss function and function f_2 is a regularizing term. We may quickly see that such a problem may be converted into the form (1.2) by introducing a second variable z and adding a constraint of the form $x = z$.

For problems in the form (1.2), people have developed the alternating direction method of multipliers (ADMM), a method that resembles the AL method, but may better utilize the particular structure of the problem. The behavior of ADMM is understood very well in the convex setting, i.e., when the functions f_1 and f_2 are convex. On the contrary, a convergence analysis of ADMM in the nonconvex setting has been lacking until relatively recently. In the second part of the dissertation, we study a machine learning application called subspace clustering. In particular, we use recent theoretical advances for ADMM to establish that it is well-suited for certain nonconvex optimization problems that arise in subspace clustering.

We summarize the two main contributions of this dissertation below:

- We propose and analyze two AL methods specifically designed to overcome a weakness of traditional AL methods as described in §2.1. The two methods share the same key feature, which is an adaptive strategy for updating the penalty parameter that is inspired by a recently proposed technique for exact penalty methods.[14–16] The adaptive procedure requires that each trial step yields a sufficiently large reduction in linearized constraint violation, thus promoting consistent progress towards constraint satisfaction. The difference between the two new methods is that one is a trust-region algorithm and the other is a line search algorithm. We also perform extensive numerical experiments to compare our adaptive AL methods with a basic AL algorithm using both a MATLAB implementation as well as a Fortran implementation that is a modification of the well-known software package LANCELOT.[17] The results of our experiments on problems from the CUTEST[18] and COPS[19] collections, as well as on optimal power flow problems[20] indicate that our adaptive algorithms outperform traditional AL methods in terms of efficiency and reliability.

- We study a machine learning task called subspace clustering. In particular, we focus on a recently proposed low-rank model for subspace clustering and propose an ADMM algorithm for solving the nonconvex optimization problem that arises from the model. We show convergence properties of the proposed ADMM algorithm and conduct numerical experiments to illustrate that the

low-rank model together with the proposed algorithm can achieve clustering accuracy that is competitive with state-of-the-art methods.

The structure of this dissertation is as follows. Chapter 2 gives a review of the basic AL method and the ADMM algorithm. In Chapter 3 and Chapter 4, we present and analyze an adaptive AL trust-region method and a linesearch variant respectively. We present extensive numerical results for our adaptive AL methods in Chapter 5. In Chapter 6, we analyze an ADMM algorithm when applied to a class of nonconvex optimization problems that naturally arise in the subspace clustering problem, and provide numerical results.

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.

Chapter 2

Background

This chapter contains background information relevant to the remaining chapters. Section 2.1 describes the mathematical setting in which we will develop our adaptive AL methods. We review the basic AL method and discuss a key weakness, which motivates the development of the adaptive methods in Chapter 3 and Chapter 4. Section 2.2 reviews the ADMM algorithm, including recent advances in its theoretical understanding for solving nonconvex problems.

2.1 Augmented Lagrangian Method

AL Methods were first studied by Hestenes[21] and Powell[22] in the late 1960s. Although overshadowed by sequential quadratic optimization and interior-point methods in recent decades, AL methods are experiencing a resurgence as interest grows in

solving extreme-scale problems. The attractive features of AL methods in this regard are that they can be implemented matrix-free[23–26] and can obtain reasonable local convergence rates under relatively weak assumptions.[12,13]

Following the settings in the seminal paper [25], which is also the basis of the successful software package LANCELOT,[17] we frame our discussion of the basic AL method in this section and the adaptive AL methods throughout Chapter 3 and Chapter 4 in the context of the following constrained problem:

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) \\ &\text{subject to} \;\; c(x) = 0, \;\; l \leq x \leq u, \end{aligned} \tag{2.1}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and constraint function $c : \mathbb{R}^n \to \mathbb{R}^m$ are assumed to be twice continuously differentiable. Elements of the lower bound vector l can be $-\infty$ and elements of the upper bound vector u can be $+\infty$. Notice that any constrained optimization problem in form (1.1) can be transformed into a problem of form (2.1) by introducing slack variables.

The Lagrangian function for problem (2.1) is defined as

$$\ell(x, y) := f(x) - c(x)^T y,$$

where $y \in \mathbb{R}^m$ is often called the dual vector for the constraint function c; the vector x is referred to as the primal vector. The vector y is also often called a Lagrange

multiplier vector, due to its importance as clarified in Definition 2.1.1 below.

Since f and c are differentiable, we denote the gradient of f and the Jacobian of c by $g(x) := \nabla f(x)$ and $J(x) := \nabla c(x)^T$ respectively. Our goal is to find a first-order solution of (2.1), which is defined as follows.

Definition 2.1.1. *A point $x^* \in \mathbb{R}^n$ is a first-order solution to problem (2.1) if there exists a Lagrange multiplier vector $y^* \in \mathbb{R}^m$ such that (x^*, y^*) is a solution to the following systems of equations:*

$$0 = F_{OPT}(x, y) := \begin{pmatrix} F_L(x, y) \\ \nabla_y \ell(x, y) \end{pmatrix} \tag{2.2}$$

where $\nabla_y \ell(x, y)$ can easily be computed to be $-c(x)$,

$$F_L(x, y) := P[x - \nabla_x \ell(x, y)] - x = P\big[x - \big(g(x) - J(x)^T y\big)\big] - x, \tag{2.3}$$

and P is a projection operator defined componentwise 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 say that a vector x is feasible for problem (2.1) if it satisfies $c(x) = 0$ and

$\ell \leq x \leq u$. We say that problem (2.1) is feasible if there exists a feasible point; otherwise, we say that problem (2.1) is infeasible.

If problem (2.1) is infeasible, then it is commonly preferred that an algorithm for solving (2.1) returns a first-order solution to the following feasibility problem:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ v(x) \ \text{ subject to } \ l \leq x \leq u, \tag{2.4}$$

where $v(x) := \frac{1}{2}\|c(x)\|_2^2$, namely a point x satisfying

$$0 = F_{\text{FEAS}}(x) := P[x - \nabla v(x)] - x = P[x - J(x)^T c(x)] - x. \tag{2.5}$$

Such points are called infeasible stationary points, which we now formally define.

Definition 2.1.2. *If* (2.5) *holds and* $v(x) > 0$*, we say that* x *is an infeasible stationary point for problem* (2.1)*, since* x *is infeasible and is a stationary point associated for the feasibility problem* (2.4)*.*

AL methods aim to find a first-order solution, or at least an infeasible stationary point for problem (2.1), by solving a sequence of bound-constrained subproblems whose objective functions are a weighted sum of the Lagrangian ℓ and the constraint violation measure v. In particular, scaling ℓ by a penalty parameter $\mu \geq 0$, each subproblem involves the minimization of the augmented Lagrangian function

$$\mathcal{L}(x, y, \mu) := \mu \ell(x, y) + v(x) = \mu(f(x) - c(x)^T y) + \tfrac{1}{2}\|c(x)\|_2^2. \tag{2.6}$$

For future reference, the gradient of the augmented Lagrangian with respect to x evaluated at (x, y, μ) is

$$\nabla_x \mathcal{L}(x, y, \mu) = \mu\big(g(x) - J(x)^T \pi(x, y, \mu)\big), \tag{2.7}$$

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

A basic AL algorithm proceeds as follows. Given values for the Lagrange multiplier vector y and penalty parameter μ, the algorithm computes

$$x(y, \mu) := \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \ \mathcal{L}(x, y, \mu) \ \text{ subject to } \ l \leq x \leq u. \tag{2.8}$$

There may be multiple solutions to the optimization problem in (2.8), or the problem may be unbounded below. However, for simplicity we assume that in (2.8) a point $x(y, \mu)$ can be computed as an approximate first-order solution. According to Definition 2.1.1, the first-order optimality condition for problem (2.8) is that x yields

$$0 = F_{\text{AL}}\big(x, y, \mu\big) := P[x - \nabla_x \mathcal{L}(x, y, \mu)] - x. \tag{2.9}$$

Inspection of the quantities in (2.3) and (2.9) reveals an important role played by the function π in (2.7). In particular, if $c(x(y, \mu)) = 0$ for $\mu > 0$, then $\pi(x(y, \mu), y, \mu) = y$ and (2.9) implies that $F_{\text{OPT}}(x(y, \mu), y) = 0$, i.e., $(x(y, \mu), y)$ is a first-order optimal solution of (2.1). For this reason, in a basic AL algorithm, if the constraint violation

at $x(y, \mu)$ is sufficiently small, then y is set to $\pi(x, y, \mu)$. Otherwise, if the constraint violation is not sufficiently small, then the penalty parameter is decreased to place a higher priority on reducing v during subsequent iterations.

Algorithm 1 outlines a complete AL algorithm. The statement of this algorithm may differ in various ways from AL methods in the literature, but we claim that the algorithmic structure is a good representation of a generic AL method.

Algorithm 1 Basic Augmented Lagrangian Algorithm

Choose constants $\{\gamma_\mu, \gamma_t\} \subset (0, 1)$.
Choose an initial primal-dual pair (x_0, y_0) and initialize $\{\mu_0, t_0\} \subset (0, \infty)$.
Set $K \leftarrow 0$ and $j \leftarrow 0$.
loop
    if $F_{\text{OPT}}(x_K, y_K) = 0$, **then**
        return the first-order optimal 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
    Compute $x(y_K, \mu_K) \leftarrow \text{argmin}_{x \in \mathbb{R}^n} \ \mathcal{L}(x, y_K, \mu_K)$ subject to $l \leq x \leq u$.
    if $\|c(x(y_K, \mu_K))\|_2 \leq t_j$, **then**
        Set $x_{K+1} \leftarrow x(y_K, \mu_K)$.
        Set $y_{K+1} \leftarrow \pi(x_{K+1}, y_K, \mu_K)$.
        Set $\mu_{K+1} \leftarrow \mu_K$.
        Set $t_{j+1} \leftarrow \gamma_t t_j$.
        Set $j \leftarrow j + 1$.
    else
        Set $x_{K+1} \leftarrow x_K$ or $x_{K+1} \leftarrow x(y_K, \mu_K)$.
        Set $y_{K+1} \leftarrow y_K$.
        Set $\mu_{K+1} \leftarrow \gamma_\mu \mu_K$.
    end if
    Set $K \leftarrow K + 1$.
end loop

The algorithms that we propose in Chapter 3 and Chapter 4 can be motivated by observing a particular drawback of Algorithm 1, namely the manner in which

the penalty parameter μ is updated. In Algorithm 1, μ is updated if and only if the **else** clause in line 18 is reached. This is deemed appropriate since after the augmented Lagrangian was minimized in line 11, the constraint violation was larger than the target value t_j; thus, the algorithm decreases μ to place a higher emphasis on reducing v in subsequent iterations. Unfortunately, a side effect of this process is that progress in the primal space based on the update in line 19 is uncertain. Indeed, in such cases, there are typically two possible outcomes. On one hand, the algorithm may set $x_{K+1} \leftarrow x_K$ so that the only result of the iteration—involving the minimization of the (nonlinear) augmented Lagrangian—is that μ is decreased. Alternatively, the algorithm may set $x_{K+1} \leftarrow x(y_K, \mu_K)$ so that progress in the primal-space may be obtained, but not necessarily; indeed, in some cases this update may be counterproductive. In this case, the only certain progress made during the iteration is the decrease of μ.

The scenario described in the previous paragraph illustrates that a basic AL algorithm may be very inefficient, especially during early iterations when the penalty parameter μ may be too large or the multiplier y is a poor estimate of the optimal multiplier vector. The methods that we propose in Chapter 3 and Chapter 4 are designed to overcome this potential inefficiency by adaptively updating the penalty parameter *during* the minimization process for the augmented Lagrangian in line 11 of Algorithm 1.

We close this section by noting that the minimization in line 11 of Algorithm 1 is

itself an iterative process, the iterations of which we refer to as "minor" iterations. This is our motivation for using K as the "major" iteration counter, so as to distinguish it from the iteration counter k used in our methods presented in Chapter 3 and Chapter 4, which are similar—in terms of computational cost—to the "minor" iterations in Algorithm 1.

2.2 Alternating Direction Method of Multipliers

ADMM was proposed in the mid 1970s by Glowinski and Marrocco[27] and Gabay and Mercier,[28] and has since been studied extensively.[29–31] It has gained much popularity recently and found numerous applications in machine learning, signal processing and networking. Applications where ADMM plays an important role include robust PCA,[32] subspace clustering,[33] sparse inverse covariance selection,[34] image restoration,[35] signal recovery,[36] wireless sensor networks,[37,38] cloud traffic management,[39] and radio access networks,[40] for example.

ADMM is most often used to solve an optimization problem that has the following

structure, which we have already seen in (1.2):

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f_1(x) + f_2(z) \\ \text{subject to} \quad & Ax + Bz = d \end{aligned} \tag{2.10}$$

where $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ are the variables over which the objective is minimized, and $A \in \mathbb{R}^{p \times n}$, $B \in \mathbb{R}^{p \times m}$, and $d \in \mathbb{R}^p$ are given quantities.

Just like the AL method in §2.1, ADMM is an iterative algorithm that utilizes the augmented Lagrangian function in its subproblems. We define the augmented Lagrangian function for problem (2.10) as

$$\mathcal{L}_\rho(x, z, y) = f_1(x) + f_2(z) - y^T(Ax + Bz - d) + \frac{\rho}{2}\|Ax + Bz - d\|_2^2, \tag{2.11}$$

where y is a Lagrange multiplier and $\rho > 0$ is a penalty parameter. Notice that here we put the penalty parameter in front of the quadratic term unlike in (2.6). However, this will not affect the algorithm as we will see from Algorithm 2 below.

The basic ADMM proceeds as follows. Given the k-th iterate (x_k, z_k, y_k) with y_k an estimate of an optimal Lagrange multiplier for the constraint $Ax + Bz - d = 0$, ADMM generates $(x_{k+1}, z_{k+1}, y_{k+1})$ by first minimizing $\mathcal{L}_\rho(x, z_k, y_k)$ with respect to x, then minimizing $\mathcal{L}_\rho(x_{k+1}, z, y_k)$ with respect to z, and finally performing a multiplier update. This procedure is shown formally in Algorithm 2.

A major difference between ADMM and a basic AL method is that in ADMM,

Algorithm 2 The basic ADMM framework

1: Choose initial values for z_0, y_0, and $\rho > 0$.
2: **while** the stopping criterion is not satisfied **do**
3: $\quad x_{k+1} = \operatorname{argmin}_x \mathcal{L}_\rho(x, z_k, y_k)$,
4: $\quad z_{k+1} = \operatorname{argmin}_z \mathcal{L}_\rho(x_{k+1}, z, y_k)$,
5: $\quad y_{k+1} = y_k - \rho(Ax_{k+1} + Bz_{k+1} - d)$.
6: **end while**

the augmented Lagrangian is reduced (in general not actually minimized) in the primal variables x and z in an alternating fashion. This design is appealing when the subproblems on lines 3 and 4 of Algorithm 2 can be easily solved, and minimizing the augmented Lagrangian jointly in x and z—as in a basic AL method—is difficult. We may also notice that the multiplier estimate y_k is updated more often in ADMM than in a basic AL algorithm in the following sense. In a basic AL algorithm, the multiplier vector is updated only after an approximate minimizer of the AL function is discovered. In ADMM, the multiplier vector is updated after a single iteration of alternating minimization is performed on the AL function. This observation, however, does not imply that ADMM will take less computational time to solve a problem than a basic AL method, or vice versa. In practice, one can observe a slower convergence rate of ADMM relative to AL in certain cases.[41]

Another difference is that the penalty parameter ρ is usually kept constant in ADMM. For one reason, a constant penalty parameter is good enough for ADMM to possess good convergence guarantees when applied to convex problems, i.e., when both f_1 and f_2 in (2.10) are convex.[42] A second reason is that no theoretically sound way of adjusting the penalty parameter is known; some heuristics have been used

with mixed success.[42] In the limited literature that studies ADMM in the nonconvex setting, the analysis is also done under the assumption that the penalty parameter is constant. Although this dissertation does not investigate the effect of a dynamic penalty parameter for ADMM, it surely is an interesting and challenging topic.

One advantage of ADMM is its ease of implementation, provided that the optimization problems in line 3 and line 4 of Algorithm 2 can easily be solved. If we look at the equations on those two lines more carefully, we will see that the x-update on line 3 is of the following form:

$$x_{k+1} = \underset{x}{\operatorname{argmin}} \; f_1(x) + \frac{\rho}{2}\|Ax - u_k\|_2^2, \tag{2.12}$$

where $u_k = -Bz_k + d + \frac{1}{\rho}y_k$. A similar observation holds for the z-update.

Form (2.12) is of interest because it can be solved efficiently in certain applications. For example, in the case where $A = I$, the righthand side of (2.12) becomes the well-known[43] proximal operator:

$$x_{k+1} = \underset{x}{\operatorname{argmin}} \; f_1(x) + \frac{\rho}{2}\|x - u_k\|_2^2. \tag{2.13}$$

When f_1 is simple enough, the righthand side of (2.13) will give a closed-form solution; see [44] for examples. Therefore, when the subproblems on lines 3 and 4 are easy to solve or have closed-form solutions, ADMM is easy to implement.

ADMM has well-established convergence results when both f_1 and f_2 are convex.[42]

When this is not the case, one has to rely on a limited set of convergence results from the literature. In a recent paper [45], the authors showed that for certain nonconvex problems (namely nonconvex consensus problems and nonconvex sharing problems), any limit point of the sequence generated by Algorithm 2 would be a first-order stationary point. We note that no claim about the existence of a limit point was given in the paper. Two other recent papers [46] and [47] studied nonconvex composite problems in certain forms and drew a stronger conclusion that any sequence generate by Algorithm 2 would converge to a first-order stationary point under more restrictive assumptions, namely the two functions f_1 and f_2 in the objective were assumed to be semi-algebraic or subanalytic. It is also worthwhile to mention that all these papers made certain assumptions on problem (2.10). For instance, one of f_1 and f_2 in the objective function was assumed to be continuously differentiable with a Lipschitz continuous gradient in all three papers. Therefore, a convergence analysis for Algorithm 2 in a more general setting is still open. Moreover, as far as I know, no result on the rate of convergence of Algorithm 2 is known in a nonconvex setting.

In the remainder of this section, we present a convergence result from 45 about ADMM in a particular nonconvex setting, which will be the basis for our application of ADMM to the subspace clustering problem in Chapter 6.

The result is framed in the context of the following nonconvex problem:

$$\begin{aligned} &\underset{x,z}{\text{minimize}} \;\; f_1(x) + f_2(z) \\ &\text{subject to} \;\; x = z \end{aligned} \tag{2.14}$$

where x and z are both in $\mathbb{R}^n$, f_1 is assumed to be convex and f_2 is assumed to be continuously differentiable. Nothing about differentiability is assumed for f_1 and nothing about convexity is assumed for f_2.

We are interested in finding a first-order solution of problem (2.14) which is defined as a pair (x^*, z^*) that satisfies:

$$\begin{aligned} 0 &\in \partial f_1(x^*) + y^*, \\ 0 &= \nabla f_2(z^*) - y^*, \\ 0 &= x^* - z^*, \end{aligned}$$

for some Lagrange multiplier $y^* \in \mathbb{R}^n$ where $\partial f_1(x)$ denotes the subdifferential of f_1 at x. The result goes as follows.

Theorem 2.2.1. *Suppose that Algorithm 2 is used to solve problem* (2.14) *and that the following three assumptions hold:*

1. *∇f_2 is Lipschitz continuous with constant L;*

2. *The penalty parameter ρ is large enough such that the z subproblem on line 4*

of Algorithm 2 is strongly convex with modulus γ, and $\rho \geq L$, $\rho\gamma > 2L^2$;

3. *The function $f(x) := f_1(x) + f_2(x)$ is bounded from below.*

Then, it follows that any limit point of the sequence $\{(x_k, z_k, y_k)\}$ generated by Algorithm 2 is a first order solution to problem (2.14).

We note that the previous result makes no claim about the existence of limit points of sequence $\{(x_k, z_k, y_k)\}$. Rather, a claim is made about such limit points when they exist. We will show, however, that for the nonconvex models encountered in subspace clustering, we can establish boundedness of the sequence $\{(x_k, z_k, y_k)\}$, which implies the existence of at least one limit point.

Chapter 3

An Adaptive AL Trust-Region Method

In this chapter we consider problem (2.1) with f and c twice continuously differentiable. We propose and analyze an adaptive AL trust-region method. In particular, we show that the algorithm is well-posed and discuss its global convergence properties.

3.1 Algorithm description

In this section, we introduce our AL trust-region method with an adaptive updating scheme for the penalty parameter. The new key idea is that at each iteration of the algorithm (similar to a "minor" iteration in Algorithm 1 as remarked in the last paragraph of §2.1), we measure the improvement towards linearized constraint satis-

faction obtained by a given trial step and compare it to that obtained by a step that solely seeks feasibility. If the former improvement is not sufficiently large compared to the latter and the current constraint violation is not sufficiently small, then the penalty parameter is decreased to place a higher emphasis on minimizing constraint violation during the current iteration.

Our strategy involves a set of easily implementable conditions designed around the following models of the constraint violation measure, Lagrangian, and augmented Lagrangian, respectively:

$$q_v(s;x) := \tfrac{1}{2}\|c(x) + J(x)s\|_2^2 \qquad \approx v(x+s); \tag{3.1}$$

$$q_\ell(s;x,y) := \ell(x,y) + \nabla_x \ell(x,y)^T s + \tfrac{1}{2} s^T \nabla^2_{xx} \ell(x,y) s \quad \approx \ell(x+s,y); \tag{3.2}$$

$$q(s;x,y,\mu) := \mu q_\ell(s;x,y) + q_v(s;x) \qquad \approx \mathcal{L}(x+s,y,\mu). \tag{3.3}$$

We remark that this approximation to the augmented Lagrangian is not the standard second-order Taylor series approximation; instead, we employ a Gauss-Newton model for the constraint violation measure.

Each iteration of our algorithm requires the computation of a trial step s_k toward minimizing the augmented Lagrangian. Ideally, this trial step will also make progress toward solving (2.1) and, in particular, toward minimizing v. To promote this behavior, we compute a step s_k that predicts a decrease in the augmented Lagrangian as well as an acceptable value of linearized constraint violation.

Whether a computed step s_k yields an acceptable value of linearized constraint violation from the current iterate x_k depends on that yielded by a *steering* step r_k, defined as an approximate solution of

$$\underset{r\in\mathbb{R}^n}{\text{minimize}}\ q_v(r;x_k)\ \text{ subject to }\ l \le x_k + r \le u,\ \|r\|_2 \le \theta_k, \tag{3.4}$$

where $\delta > 0$ is a constant and both

$$\theta_k := \theta(x_k,\delta_k) := \min\{\delta_k, \delta\|F_{\text{FEAS}}(x_k)\|_2\} \ge 0 \tag{3.5}$$

and $\delta_k > 0$ are set dynamically within our algorithm. (Note that a consequence of this choice of trust-region radius θ_k in (3.4) is that it approaches zero as the algorithm approaches stationary points of the constraint violation measure.[48] This keeps the steering step from being too large relative to the progress that can be made toward minimizing v.) Problem (3.4) is, in fact, a quadratic optimization problem if we used the ℓ_∞-norm to define the trust-region constraint. Nonetheless, our algorithm uses the ℓ_2-norm and allows for inexact solutions to this subproblem that still ensure convergence since we are interested in matrix-free implementations of our methods; see (3.11a). To this end, we compute a Cauchy step for subproblem (3.4) as

$$\overline{r}_k := \overline{r}(x_k,\theta_k) := P[x_k - \overline{\beta}_k J_k^T c_k] - x_k \tag{3.6}$$

such that $\overline{r}_k$ satisfies

$$\Delta q_v(\overline{r}_k; x_k) := q_v(0; x_k) - q_v(\overline{r}_k; x_k) \geq -\varepsilon_r \overline{r}_k^T J_k^T c_k \quad \text{and} \quad \|\overline{r}_k\|_2 \leq \theta_k \tag{3.7}$$

for some $\overline{\beta}_k := \overline{\beta}(x_k, \theta_k)$ and $\varepsilon_r \in (0, 1)$. Appropriate values for $\overline{\beta}_k$, $\overline{r}_k$, and the auxiliary nonnegative scalar quantities ε_k and Γ_k (to be used shortly) may be computed from Algorithm 3.

Algorithm 3 Cauchy step computation for the feasibility subproblem (3.4)

procedure CAUCHY_FEASIBILITY(x_k, θ_k)
  restrictions : $\theta_k \geq 0$.
  available constants : $\{\varepsilon_r, \gamma\} \subset (0, 1)$.
  Let l_k be the smallest nonnegative integer so that $\|P[x_k - \gamma^{l_k} J_k^T c_k] - x_k\|_2 \leq \theta_k$.
  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 (3.7) **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

The predicted reduction in the constraint violation from x_k yielded by $\overline{r}_k$ as measured by $\Delta q_v(\overline{r}_k; x_k)$ is guaranteed to be positive at any x_k that is not first-order critical for v under the bound constraints; see part (i) of Lemma 3.2.5. The reduction $\Delta q_v(r_k; x_k)$ is defined similarly for the steering step r_k, whose computation will be discussed later in this section.

With the Cauchy step for our steering problem computed, we proceed to identify a new penalty parameter μ_k and trial step s_k that satisfy certain properties. Specifically, the step s_k is defined as an approximate solution of the following quadratic trust-region subproblem:

$$\underset{s\in\mathbb{R}^n}{\text{minimize}}\ q(s; x_k, y_k, \mu_k)\ \text{ subject to }\ l \le x_k + s \le u,\ \|s\|_2 \le \Theta_k, \tag{3.8}$$

where

$$\Theta_k := \Theta(x_k, y_k, \mu_k, \delta_k, \Gamma_k) = \Gamma_k \min\{\delta_k, \delta\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2\} \ge 0 \tag{3.9}$$

with $\Gamma_k > 1$ returned from Algorithm 3. Similar to (3.4), this definition of the trust-region radius involves the first-order optimality measure F_{AL} for minimizing $\mathcal{L}(\cdot, y_k, \mu_k)$. This choice ensures that the trust-region radius Θ_k is driven to zero as first-order minimizers of $\mathcal{L}(\cdot, y_k, \mu_k)$ are approached. Moreover, since in practice δ is chosen to be large, the term $\delta\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2$ will not impede superlinear convergence when the inverse of the Hessian matrix is smaller in norm than δ in the neighborhood of solutions.

Problem (3.8) is also a quadratic optimization problem if we used the ℓ_∞-norm to define the trust-region constraint. Nonetheless, our algorithm uses the ℓ_2-norm and allows for inexact solutions to this subproblem that still ensure convergence; see

(3.11a). We define the Cauchy step for problem (3.8) as

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

such that $\overline{s}_k$ yields

$$\begin{aligned} \Delta q(\overline{s}_k; x_k, y_k, \mu_k) &:= q(0; x_k, y_k, \mu_k) - q(\overline{s}_k; x_k, y_k, \mu_k) \\ &\geq -\frac{(\varepsilon_k + \varepsilon_r)}{2} \overline{s}_k^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k) \quad \text{and} \quad \|\overline{s}_k\|_2 \leq \Theta_k \end{aligned} \tag{3.10}$$

for some $\overline{\alpha}_k = \overline{\alpha}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$, where $\varepsilon_k \geq 0$ is returned from Algorithm 3. Appropriate values for $\overline{\alpha}_k$ and $\overline{s}_k$ may be computed from Algorithm 4. (The importance of using Γ_k in (3.9) and ε_k in (3.10) may be seen in the proofs of Lemmas 3.2.3 and 3.2.4 in §3.2.)

Algorithm 4 Cauchy step computation for the AL subproblem (3.8).

procedure CAUCHY_AL($x_k, y_k, \mu_k, \Theta_k, \varepsilon_k$)
  restrictions : $\mu_k > 0$, $\Theta_k \geq 0$, and $\varepsilon_k \geq 0$.
  available constants : $\{\varepsilon_r, \gamma\} \subset (0, 1)$.
  Set $\overline{\alpha}_k \leftarrow 1$ and $\overline{s}_k \leftarrow P[x_k - \overline{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)] - x_k$.
  while (3.10) is not satisfied **do**
    Set $\overline{\alpha}_k \leftarrow \gamma \overline{\alpha}_k$ and $\overline{s}_k \leftarrow P[x_k - \overline{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)] - x_k$.
  end while
  return : $(\overline{\alpha}_k, \overline{s}_k)$
end procedure

The predicted reduction in $\mathcal{L}(\cdot, y_k, \mu_k)$ from x_k yielded by the step $\overline{s}_k$ and measured by $\Delta q(\overline{s}_k; x_k, y_k, \mu_k)$ is guaranteed to be positive at any x_k that is not first-order critical for $\mathcal{L}(\cdot, y_k, \mu_k)$ under the bound constraints; see part (ii) of Lemma 3.2.5 for a

more precise lower bound for this predicted change. The reduction $\Delta q(s_k; x_k, y_k, \mu_k)$ is defined similarly for the trial step s_k.

We now describe the kth iteration of our algorithm, specified as Algorithm 5 on page 29. Let (x_k, y_k) be the current primal-dual iterate. We begin by checking whether (x_k, y_k) is a first-order optimal point for (2.1) as given by Definition 2.1.1 or if x_k is an infeasible stationary point as given by Definition 2.1.2, and terminate in either case. Otherwise, we enter the **while** loop in line 11 to obtain a value for the penalty parameter for which $F_{\text{AL}}(x_k, y_k, \mu_k) \neq 0$; recall (2.9). This is appropriate as the purpose of each iteration is to compute a step towards a bound-constrained minimizer of the augmented Lagrangian $\mathcal{L}(\cdot, y_k, \mu_k)$, and if $F_{\text{AL}}(x_k, y_k, \mu_k) = 0$, then no feasible descent directions for $\mathcal{L}(\cdot, y_k, \mu_k)$ from x_k exist. (Lemma 3.2.2 shows that this **while** loop terminates finitely.) Next, we enter a **while** loop on line 19 that recovers an approximate solution r_k to problem (3.4) and an approximate solution s_k to problem (3.8) that satisfy

$$\Delta q(s_k; x_k, y_k, \mu_k) \geq \kappa_1 \Delta q(\bar{s}_k; x_k, y_k, \mu_k) > 0,\ l \leq x_k + s_k \leq u,\ \|s_k\|_2 \leq \Theta_k, \quad (3.11a)$$

$$\Delta q_v(r_k; x_k) \geq \kappa_2 \Delta q_v(\bar{r}_k; x_k),\ l \leq x_k + r_k \leq u,\ \|r_k\|_2 \leq \theta_k, \quad (3.11b)$$

$$\text{and}\ \ \Delta q_v(s_k; x_k) \geq \min\{\kappa_3 \Delta q_v(r_k; x_k), v_k - \tfrac{1}{2}(\kappa_t t_j)^2\}, \quad (3.11c)$$

where $\{\kappa_1, \kappa_2, \kappa_3, \kappa_t\} \subset (0, 1)$ and the quantity $t_j > 0$ represents the jth target for the constraint violation. At this stage of the algorithm, there are many vectors r_k and s_k

that satisfy (3.11), but for our purposes we simply prove that they are satisfied for $r_k = \overline{r}_k$ and $s_k = \overline{s}_k$; see Theorem 3.2.1.

The conditions in (3.11) can be motivated as follows. Conditions (3.11a) and (3.11b) ensure that the trial step s_k and steering step r_k yield nontrivial decreases in the models of the augmented Lagrangian and the constraint violation, respectively, compared to their Cauchy points. The motivation for condition (3.11c) is more complex as it involves a minimum of two values on the right-hand side, but this condition is critical as it ensures that the reduction in the constraint violation model is sufficiently large for the trial step. The first quantity on the right-hand side, if it were the minimum of the two, would require the decrease in the model q_v yielded by s_k to be a fraction of that obtained by the steering step r_k; see [14, 15] for similar conditions enforced in exact penalty methods. The second quantity is the difference between the current constraint violation and a measure involving a fraction of the target value $t_j > 0$. Note that this second term allows the minimum to be negative. Therefore, this condition allows for the trial step s_k to predict an increase in the constraint violation, but only if the current constraint violation is sufficiently within the target value t_j. It is worthwhile to note that one may consider allowing the penalty parameter to increase as long as the resulting trial step satisfies conditions (3.11a)–(3.11c) and the parameter eventually settles down at a small enough value to ensure that constraint violation is minimized. However, as this only would be a heuristic and not theoretically interesting, we ignore this possibility and simply have the parameter decrease

monotonically. We remark that the computation of r_k requires extra effort beyond that for computing s_k, but this expense is minor since r_k can be computed in parallel with s_k and must only satisfy the Cauchy decrease condition (3.11b) for (3.4).

With the trial step s_k in hand, we proceed to compute the ratio

$$\rho_k \leftarrow \frac{\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_k + s_k, y_k, \mu_k)}{\Delta q(s_k; x_k, y_k, \mu_k)} \tag{3.12}$$

of actual-to-predicted decrease in $\mathcal{L}(\cdot, y_k, \mu_k)$. Since $\Delta q(s_k; x_k, y_k, \mu_k)$ is positive by (3.11a), it follows that if $\rho_k \geq \eta_s$ for $\eta_s \in (0, 1)$, then the augmented Lagrangian has been sufficiently reduced. In such cases, we accept $x_k + s_k$ as the next iterate. Moreover, if we find that $\rho_k \geq \eta_{vs}$ for $\eta_{vs} \in (\eta_s, 1)$, then our choice of trust-region radius may have been overly cautious so we multiply the upper bound for the trust-region radius (i.e., δ_k) by $\Gamma_\delta > 1$. If $\rho_k < \eta_s$, then the trust-region radius may have been too large, so we counter this by multiplying the upper bound by $\gamma_\delta \in (0, 1)$.

Next we determine how to define our next multiplier vector y_{k+1}. The first condition that we check is whether the constraint violation at x_{k+1} is sufficiently small compared to t_j. If this requirement is met, then we may compute any prospective multiplier estimate $\widehat{y}_{k+1}$ that satisfies

$$\|F_{\text{L}}(x_{k+1}, \widehat{y}_{k+1})\|_2 \leq \min\left\{\|F_{\text{L}}(x_{k+1}, y_k)\|_2, \|F_{\text{L}}(x_{k+1}, \pi(x_{k+1}, y_k, \mu_k))\|_2\right\}. \tag{3.13}$$

Computing $\widehat{y}_{k+1}$ as an approximate least-squares multiplier estimate from an associ-

Algorithm 5 Adaptive Augmented Lagrangian Trust-Region Algorithm

Choose $\{\gamma, \gamma_\mu, \gamma_t, \gamma_T, \gamma_\delta, \kappa_1, \kappa_2, \kappa_3, \varepsilon_r, \kappa_t, \eta_s, \eta_{vs}\} \subset (0,1)$, $\{\delta, \delta_R, \epsilon, Y\} \subset (0,\infty)$, $\Gamma_\delta > 1$ such that $\eta_{vs} \geq \eta_s$.
Choose initial primal-dual pair (x_0, y_0) and initialize $\{\mu_0, \delta_0, t_0, t_1, T_1, Y_1\} \subset (0,\infty)$ such that $Y_1 \geq Y$ and $\|y_0\|_2 \leq Y_1$.
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 optimal 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
    Define θ_k by (3.5).
    Use Algorithm 3 to compute $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) = \text{CAUCHY\_FEASIBILITY}(x_k, \theta_k)$.
    Define Θ_k by (3.9).
    Use Algorithm 4 to compute $(\overline{\alpha}_k, \overline{s}_k) = \text{CAUCHY\_AL}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
    Compute approx. solutions r_k/s_k to (3.4)/(3.8) satisfying (3.11a)–(3.11b).
    while (3.11c) is not satisfied or $F_{\text{AL}}(x_k, y_k, \mu_k) = 0$, **do**
        Set $\mu_k \leftarrow \gamma_\mu \mu_k$ and define Θ_k by (3.9).
        Use Algorithm 4 to compute $(\overline{\alpha}_k, \overline{s}_k) = \text{CAUCHY\_AL}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
        Compute an approximate solution s_k to (3.8) satisfying (3.11a).
    end while
    Compute ρ_k from (3.12).
    if $\rho_k \geq \eta_{vs}$, **then**
        Set $x_{k+1} \leftarrow x_k + s_k$ and $\delta_{k+1} \leftarrow \max\{\delta_R, \Gamma_\delta \delta_k\}$. ▷ very successful iteration
    else if $\rho_k \geq \eta_s$, **then**
        Set $x_{k+1} \leftarrow x_k + s_k$ and $\delta_{k+1} \leftarrow \max\{\delta_R, \delta_k\}$. ▷ successful iteration
    else
        Set $x_{k+1} \leftarrow x_k$ and $\delta_{k+1} \leftarrow \gamma_\delta \delta_k$. ▷ unsuccessful iteration
    end if
    if $\|c_{k+1}\|_2 \leq t_j$, **then**
        Compute any $\widehat{y}_{k+1}$ satisfying (3.13).
        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 $T_{j+1} \leftarrow \gamma_T \min\{1, \mu_k\} T_j$.
            Set y_{k+1} from (3.14) where α_y satisfies (3.15).
            Set $j \leftarrow j+1$.

39: **else**
40: Set $y_{k+1} \leftarrow y_k$.
41: **end if**
42: **else**
43: Set $y_{k+1} \leftarrow y_k$.
44: **end if**
45: Set $\mu_{k+1} \leftarrow \mu_k$ and then set $k \leftarrow k + 1$.
46: **end loop**

ated linear optimization problem or simply from $\pi(x_{k+1}, y_k, \mu_k)$ or y_k depending on the minimum of the right hand side of (3.13) are all viable options, but for flexibility in the statement of our algorithm we simply enforce (3.13). With it being satisfied, we then check 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 a target value $T_j > 0$. If this condition is satisfied, we choose new target values $t_{j+1} < t_j$ and $T_{j+1} < T_j$, and then set $Y_{j+1} \geq Y_j$ and

$$y_{k+1} \leftarrow (1 - \alpha_y)y_k + \alpha_y \widehat{y}_{k+1}, \tag{3.14}$$

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

$$\|(1 - \alpha_y)y_k + \alpha_y \widehat{y}_{k+1}\|_2 \leq Y_{j+1}. \tag{3.15}$$

This updating procedure is well-defined since the choice $\alpha_y \leftarrow 0$ results in $y_{k+1} \leftarrow y_k$, which at least means that (3.15) is satisfiable by this α_y. On the other-hand, i.e., when the aforementioned condition is not satisfied, we simply set $y_{k+1} \leftarrow y_k$.

For future reference, we define the subset of iterations where line 35 of Algorithm 5

is reached as

$$\mathcal{Y} := \left\{ k_j : \|c_{k_j}\|_2 \leq t_j \text{ and } \min\{\|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2, \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2\} \leq T_j \right\}. \tag{3.16}$$

We conclude this section by noting that, in practice, lines 5 and 8 in Algorithm 5 should be replaced by practical conditions that include positive stopping tolerances. For example, see the implementation details given in §5.1.

3.2 Well-posedness

We prove that Algorithm 5 is well-posed—i.e., either the algorithm will terminate finitely or will produce an infinite sequence $\{(x_k, y_k, \mu_k)\}_{k\geq 0}$ of iterates—under the following assumption. This assumption is assumed throughout this section and is therefore not stated explicitly in each result.

Assumption 3.2.1. *At a given x_k, the objective function f and constraint function c are both twice-continuously differentiable.*

Well-posedness in our context first requires that the **while** loop that begins at line 11 of Algorithm 5 terminates finitely. The proof of this fact requires the following simple result.

Lemma 3.2.1. *Suppose $l \leq x \leq u$ and let v be any vector in $\mathbb{R}^n$. If there exists a scalar $\xi_s > 0$ such that $P[x - \xi_s v] = x$, then $P[x - \xi v] = x$ for all $\xi > 0$.*

Proof. Since $l \leq x \leq u$, it follows the definition of the projection operator P and the facts that $\xi_s > 0$ and $P[x - \xi_s v] = x$ that

$$v_i \geq 0 \text{ if } x_i = l_i; \quad v_i \leq 0 \text{ if } x_i = u_i; \quad \text{and } v_i = 0 \text{ otherwise.}$$

It follows that $P[x - \xi v] = x$ for all $\xi > 0$, as desired. □

Lemma 3.2.2. *If line 11 is reached, $F_{AL}(x_k, y_k, \mu) \neq 0$ for all sufficiently small $\mu > 0$.*

Proof. Suppose that line 11 is reached and, to reach a contradiction, suppose also that there exists an infinite positive sequence $\{\xi_h\}_{h\geq 0}$ such that $\xi_h \to 0$ and

$$F_{\text{AL}}(x_k, y_k, \xi_h) = P\big[x_k - \xi_h(g_k - J_k^T y_k) - J_k^T c_k\big] - x_k = 0 \text{ for all } h \geq 0. \tag{3.17}$$

It follows from (3.17) and the fact that $\xi_h \to 0$ that

$$F_{\text{FEAS}}(x_k) = P[x_k - J_k^T c_k] - x_k = 0.$$

If $c_k \neq 0$, then Algorithm 5 would have terminated in line 9; hence, since line 11 is reached, we must have $c_k = 0$. We then may conclude from (3.17), the fact that $\{\xi_h\}$ is positive for all h, and Lemma 3.2.1 that $F_{\text{L}}(x_k, y_k) = 0$. Combining this with (2.2) and the fact that $c_k = 0$, it follows that $F_{\text{OPT}}(x_k, y_k) = 0$ so that (x_k, y_k) is a first-order optimal point for (2.1). Under these conditions, Algorithm 5 would have terminated in line 6. Hence, we have a contradiction to the existence of the sequence $\{\xi_h\}$. □

We now show that the Cauchy step computations given by Algorithms 3 and 4 are well defined when called in steps 15, 17, and 21 of Algorithm 5.

Lemma 3.2.3. *The following hold true for Algorithm 5:*

(i) The computation of $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k)$ *in step 15 is well defined and yields the values* $\Gamma_k \in (1, 2]$ *and* $\varepsilon_k \in [0, \varepsilon_r)$.

(ii) The computation of $(\overline{\alpha}_k, \overline{s}_k)$ *in steps 17 and 21 is well defined.*

Proof. We first prove part (i). Consider the call to Algorithm 3 made during step 15 of Algorithm 5. If $\theta_k = 0$, then, since $\delta_k > 0$ by construction, we must have $F_{\text{FEAS}}(x_k) = 0$, which in turn implies that Algorithm 3 trivially computes $l_k = 0$, $\Gamma_k = 2$, $\overline{\beta}_k = 1$, and $\varepsilon_k = 0$. In this case, (i) clearly holds, so now let us suppose that $\theta_k > 0$. If $l_k = 0$, then $\Gamma_k = 2$; otherwise, if $l_k > 0$, then it follows that either $\Gamma_k = 2$ or

$$2 > \Gamma_k = \tfrac{1}{2}\left(1 + \frac{\|P\left[x_k - \gamma^{l_k - 1} J_k^T c_k\right] - x_k\|_2}{\theta_k}\right) > \tfrac{1}{2}\left(1 + \frac{\theta_k}{\theta_k}\right) = 1.$$

Next, the **while** loop at line 11 of Algorithm 3 terminates finitely as shown by [49, Theorem 4.2] since $\varepsilon_r \in (0, 1)$. The fact that $\varepsilon_k \in [0, \varepsilon_r)$ holds since $\varepsilon_r \in (0, 1)$ by choice, ε_k is initialized to zero in Algorithm 3, and every time ε_k is updated we have $-\Delta q_v(\overline{r}_k; x_k)/\overline{r}_k^T J_k^T c_k < \varepsilon_r$ (by the condition of the **while** loop).

Now consider part (ii). Regardless of how step 17 or 21 is reached in Algorithm 5, we have that $\mu_k > 0$ and $\Theta_k \geq 0$ by construction, and from part (i) we have that $\Gamma_k \in (1, 2]$ and $\varepsilon_k \in [0, \varepsilon_r)$. If $\Theta_k = 0$, then, since $\delta_k > 0$ by construction, it follows

that $F_{\text{AL}}(x_k, y_k, \mu_k) = 0$, and therefore Algorithm 4 terminates with $\overline{\alpha}_k = 1$ and $\overline{s}_k = 0$. Thus, we may continue supposing that $\Theta_k > 0$. Now, using the fact that

$$0 < \varepsilon_r/2 \leq (\varepsilon_k + \varepsilon_r)/2 < \varepsilon_r < 1$$

and [49, Theorem 4.2] we may say that Algorithm 4 will terminate finitely with $(\overline{\alpha}_k, \overline{s}_k)$ satisfying (3.10). □

The following result illustrates critical relationships between the quadratic models q_v and q as $\mu \to 0$.

Lemma 3.2.4. *Let* $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) \leftarrow$ CAUCHY_FEASIBILITY(x_k, θ_k) *with* θ_k *defined by* (3.5) *and let* $(\overline{\alpha}_k(\mu), \overline{s}_k(\mu)) \leftarrow$ CAUCHY_AL$(x_k, y_k, \mu, \Theta_k(\mu), \varepsilon_k)$ *with* $\Theta_k(\mu) := \Gamma_k \min\{\delta_k, \delta \|F_{AL}(x_k, y_k, \mu)\|_2\}$ *(see* (3.9)*). Then, the following hold true:*

$$\lim_{\mu \to 0} \left(\max_{\|s\|_2 \leq 2\theta_k} |q(s; x_k, y_k, \mu) - q_v(s; x_k)| \right) = 0, \tag{3.18a}$$

$$\lim_{\mu \to 0} \nabla_x \mathcal{L}(x_k, y_k, \mu) = J_k^T c_k, \tag{3.18b}$$

$$\lim_{\mu \to 0} \overline{s}_k(\mu) = \overline{r}_k, \tag{3.18c}$$

$$\textit{and} \quad \lim_{\mu \to 0} \Delta q_v(\overline{s}_k(\mu); x_k) = \Delta q_v(\overline{r}_k; x_k). \tag{3.18d}$$

Proof. Since x_k and y_k are fixed, for the purposes of this proof we drop them from all function dependencies. From the definitions of q and q_v, it follows that for some

$M > 0$ independent of μ we have

$$\max_{\|s\|_2 \leq 2\theta_k} |q(s;\mu) - q_v(s)| = \mu \max_{\|s\|_2 \leq 2\delta_k} |q_\ell(s)| \leq \mu M.$$

Hence, (3.18a) follows. Similarly, we have

$$\nabla_x \mathcal{L}(\mu) - J_k^T c_k = \mu(g_k - J_k^T y_k),$$

from which it is clear that (3.18b) holds.

We now show (3.18c) by considering two cases. We emphasize that throughout these two arguments all quantities in Algorithm 3 are unaffected by μ, so the reader can consider them as fixed.

Case 1: Suppose that $F_{\text{FEAS}}(x_k) = 0$. This implies that $\theta_k = \min\{\delta_k, \delta\|F_{\text{FEAS}}(x_k)\|_2\} = 0$, so that $\overline{r}_k = 0$ and $\Delta q_v(\overline{r}_k) = 0$. Moreover, from (3.18b) we have $\Theta_k(\mu) \to 0$ as $\mu \to 0$, which means that $\overline{s}_k(\mu) \to 0 = \overline{r}_k$.

Case 2: Suppose $F_{\text{FEAS}}(x_k) \neq 0$. We find it useful to define the functions

$$r_k(l) = P[x_k - \gamma^l J_k^T c_k] - x_k \quad \text{and} \quad s_k(l,\mu) = P[x_k - \gamma^l \nabla_x \mathcal{L}(\mu)] - x_k$$

for any integer $l \geq 0$ and scalar $\mu > 0$. We also let $l_\beta \geq 0$ be the integer such that $\overline{\beta}_k = \gamma^{l_\beta}$, which implies

$$\overline{r}_k = r_k(l_\beta). \tag{3.19}$$

It follows from (3.18b) that

$$\lim_{\mu\to 0} s_k(l,\mu) = r_k(l) \;\text{ for any } l \geq 0 \qquad \text{and} \qquad \lim_{\mu\to 0} s_k(l_\beta,\mu) = r_k(l_\beta) = \overline{r}_k. \tag{3.20}$$

Therefore, to show (3.18c), it suffices to prove that

$$\overline{s}_k(\mu) = s_k(l_\beta,\mu) \quad \text{for all } \mu > 0 \text{ sufficiently small.} \tag{3.21}$$

Since the Cauchy computation for $\overline{s}_k(\mu)$ computes a nonnegative integer $l_{\alpha,\mu}$ so that

$$\overline{s}_k(\mu) = P\big[x_k - \gamma^{l_{\alpha,\mu}} \nabla_x \mathcal{L}(\mu)\big] - x_k,$$

to prove (3.21) it suffices to show that $l_{\alpha,\mu} = l_\beta$ for all $\mu > 0$ sufficiently small. Before proving this result, however, we show that

$$\min\big(l_\beta, l_{\alpha,\mu}\big) \geq l_k \quad \text{for all } \mu > 0 \text{ sufficiently small,} \tag{3.22}$$

where l_k is computed in Algorithm 3. If $l_k = 0$, then (3.22) holds trivially. Thus, let us suppose that $l_k > 0$. We may first observe that the inequality $l_\beta \geq l_k$ holds by construction of Algorithm 3. Also, it follows from the definition of $\Theta_k(\mu)$, (3.18b), the definition of Γ_k, $\theta_k > 0$, and the fact that $\|P[x_k - \gamma^{l_k-1} J_k^T c_k\big] - x_k\|_2 > \theta_k$ due to

the choice of l_k in Algorithm 3, that

$$\begin{aligned}
\lim_{\mu\to 0}\Theta_k(\mu) &= \lim_{\mu\to 0}\Gamma_k \min(\delta_k, \delta\|F_{\text{AL}}(x_k, y_k, \mu)\|_2) \\
&= \Gamma_k \min(\delta_k, \delta\|F_{\text{FEAS}}(x_k)\|_2) \\
&= \Gamma_k\theta_k \\
&= \min\left[2, \tfrac{1}{2}\left(1 + \frac{\|P[x_k - \gamma^{l_k-1}J_k^Tc_k] - x_k\|_2}{\theta_k}\right)\right]\theta_k \\
&= \min\left[2\theta_k, \tfrac{1}{2}\left(\theta_k + \|P[x_k - \gamma^{l_k-1}J_k^Tc_k] - x_k\|_2\right)\right] \\
&\in \left(\theta_k,\ \|P[x_k - \gamma^{l_k-1}J_k^Tc_k] - x_k\|_2\right).
\end{aligned}$$

Using this and (3.18b), we may observe that

$$\lim_{\mu\to 0}\|P[x_k - \gamma^{l_k-1}\nabla_x\mathcal{L}(\mu)] - x_k\|_2 = \|P[x_k - \gamma^{l_k-1}J_k^Tc_k] - x_k\|_2 > \Theta_k(\mu)$$

for all $\mu > 0$ sufficiently small, which shows that $l_{\alpha,\mu} \geq l_k$ for all $\mu > 0$ sufficiently small and, consequently, that (3.22) again holds.

We now proceed to prove that $l_{\alpha,\mu} = l_\beta$ for all $\mu > 0$ sufficiently small. It follows from the definition of l_β above, (3.19), the structure of Algorithm 3, definition of ε_k, and part (i) of Lemma 3.2.3 that

$$-\frac{\Delta q_v(\overline{r}_k)}{\overline{r}_k^TJ_k^Tc_k} = -\frac{\Delta q_v\big(r_k(l_\beta)\big)}{r_k(l_\beta)^TJ_k^Tc_k} \geq \epsilon_r \quad \text{and} \quad -\frac{\Delta q_v\big(r_k(l)\big)}{r_k(l)^TJ_k^Tc_k} \leq \epsilon_k < \epsilon_r \tag{3.23}$$

for all integers $l_k \leq l < l_\beta$. (Note that [50, Theorem 12.1.4] shows that all denominators in (3.23) are negative.) It follows from (3.18b), (3.20), (3.18a), (3.23), and part (i) of Lemma 3.2.3 that

$$\lim_{\mu \to 0} -\frac{\Delta q\big(s_k(l_\beta, \mu)\big)}{s_k(l_\beta, \mu)^T \nabla_x \mathcal{L}(\mu)} = -\frac{\Delta q_v(\overline{r}_k)}{\overline{r}_k^T J_k^T c_k} \geq \varepsilon_r > \frac{\varepsilon_k + \varepsilon_r}{2} \tag{3.24}$$

and

$$\lim_{\mu \to 0} -\frac{\Delta q\big(s_k(l, \mu)\big)}{s_k(l, \mu)^T \nabla_x \mathcal{L}(\mu)} = -\frac{\Delta q_v\big(r_k(l)\big)}{r_k(l)^T J_k^T c_k} \leq \varepsilon_k < \frac{\varepsilon_k + \varepsilon_r}{2} \tag{3.25}$$

for all integers $l_k \leq l < l_\beta$. It now follows from (3.22), (3.24), (3.25), and (3.10) that $l_{\alpha,\mu} = l_\beta$ for all $\mu > 0$ sufficiently small, which proves (3.18c).

Finally, notice that (3.18d) follows from (3.18c) and continuity of the model q_v. □

To show that Algorithm 5 is well-posed, we also need the following results.

Lemma 3.2.5. *Let Ω be any value such that*

$$\Omega \geq \max\{\|\mu_k \nabla^2_{xx} \ell(x_k, y_k) + J_k^T J_k\|_2, \|J_k^T J_k\|_2\}. \tag{3.26}$$

Then, the following hold true:

(i) For some $\kappa_4 \in (0, 1)$, the Cauchy step for subproblem (3.4) yields

$$\Delta q_v(\overline{r}_k; x_k) \geq \kappa_4 \|F_{FEAS}(x_k)\|_2 \min\left\{\theta_k, \frac{1}{1+\Omega}\|F_{FEAS}(x_k)\|_2\right\}. \tag{3.27}$$

(ii) For some $\kappa_5 \in (0,1)$, the Cauchy step for subproblem (3.8) yields

$$\Delta q(\overline{s}_k; x_k, y_k, \mu_k) \geq \kappa_5 \|F_{AL}(x_k, y_k, \mu_k)\|_2 \min\left\{\Theta_k, \frac{1}{1+\Omega}\|F_{AL}(x_k, y_k, \mu_k)\|_2\right\}. \tag{3.28}$$

Proof. We first show (3.27). We know from [49, Theorem 4.4] and (3.10) that

$$\Delta q_v(\overline{r}_k; x_k) \geq \epsilon_r \bar{\kappa}_4 \|F_{\text{FEAS}}(x_k)\|_2 \min\left\{\theta_k, \frac{1}{\Sigma_k}\|F_{\text{FEAS}}(x_k)\|_2\right\}$$

for some $\bar{\kappa}_4 \in (0,1)$ with $\Sigma_k := 1 + \sup\{|\omega_k(r)| : 0 < \|r\|_2 \leq \theta_k\}$ and

$$\omega_k(r) := \frac{-\Delta q_v(r; x_k) - r^T J_k^T c_k}{\|r\|_2^2}.$$

By rewriting $\omega_k(r)$ and using (3.26), the Cauchy-Schwartz inequality, and standard norm inequalities, we have that

$$\omega_k(r) = \frac{r^T J_k^T J_k r}{2\|r\|_2^2} \leq \Omega.$$

Therefore, $\Sigma_k \leq 1 + \Omega$ and (3.27) follows immediately with $\kappa_4 := \epsilon_r \bar{\kappa}_4$.

Now we show (3.28). We know from [49, Theorem 4.4] and (3.10) that

$$\Delta q(\overline{s}_k; x_k, y_k, \mu_k) \geq \frac{\epsilon_k + \epsilon_r}{2} \bar{\kappa}_5 \|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2 \min\left\{\Theta_k, \frac{1}{\bar{\Sigma}_k}\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2\right\}$$

for some $\bar{\kappa}_5 \in (0,1)$ with $\bar{\Sigma}_k := 1 + \sup\{|\bar{\omega}_k(s)| : 0 < \|s\|_2 \leq \Theta_k\}$ and

$$\bar{\omega}_k(s) := \frac{-\Delta q(s; x_k, y_k, \mu_k) - s^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k)}{\|s\|_2^2}.$$

By rewriting $\bar{\omega}_k(s)$ and using (3.26), we have that

$$\begin{aligned}\bar{\omega}_k(s) &= \frac{\mu_k s^T \nabla^2_{xx} \ell(x_k, y_k, \mu_k) s + s^T J_k^T J_k s}{2\|s\|_2^2} \\ &\leq \frac{\|\mu_k \nabla^2_{xx} \ell(x_k, y_k, \mu_k) + J_k^T J_k\|_2 \|s\|_2^2}{2\|s\|_2^2} \leq \Omega.\end{aligned}$$

Thus, $\bar{\Sigma}_k \leq 1 + \Omega$, which means that (3.28) follows immediately with the definition $\kappa_5 := \frac{1}{2}\epsilon_r \bar{\kappa}_5 \leq \frac{1}{2}(\varepsilon_k + \varepsilon_r)\bar{\kappa}_5$. □

In the following theorem, we combine the previous lemmas to prove that Algorithm 5 is well-posed.

Theorem 3.2.1. *The kth iteration of Algorithm 5 is well-posed. That is, either the algorithm will terminate in line 6 or 9, 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 (3.11) will be satisfied, in which case $(x_{k+1}, y_{k+1}, \mu_{k+1})$ will be computed.*

Proof. If in the kth iteration Algorithm 5 terminates in line 6 or 9, then there is nothing to prove. Therefore, for the remainder of the proof, we assume that line 11

is reached. Lemma 3.2.2 then ensures that

$$F_{\text{AL}}(x_k, y_k, \mu) \neq 0 \text{ for all } \mu > 0 \text{ sufficiently small.} \tag{3.29}$$

Consequently, the **while** loop in line 11 will terminate for a sufficiently small $\mu_k > 0$.

By construction, conditions (3.11a) and (3.11b) are satisfied for any $\mu_k > 0$ by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. Thus, all that remains is to show that for a sufficiently small $\mu_k > 0$, (3.11c) is also satisfied by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. From (3.18d), we have that

$$\lim_{\mu \to 0} \Delta q_v(s_k; x_k) = \lim_{\mu \to 0} \Delta q_v(\overline{s}_k; x_k) = \Delta q_v(\overline{r}_k; x_k) = \Delta q_v(r_k; x_k). \tag{3.30}$$

If $\Delta q_v(r_k; x_k) > 0$, then (3.30) implies that (3.11c) will be satisfied for sufficiently small $\mu_k > 0$. On the other hand, suppose

$$\Delta q_v(r_k; x_k) = \Delta q_v(\overline{r}_k; x_k) = 0, \tag{3.31}$$

which along with (3.27) and the definitions of θ_k and $\delta_k > 0$, must mean that $F_{\text{FEAS}}(x_k) = 0$. If $c_k \neq 0$, then Algorithm 5 would have terminated in line 9 and, therefore, we must have $c_k = 0$. This and (3.31) imply that

$$\min\{\kappa_3 \Delta q_v(r_k; x_k), v_k - \tfrac{1}{2}(\kappa_t t_j)^2\} = -\tfrac{1}{2}(\kappa_t t_j)^2 < 0 \tag{3.32}$$

since $t_j > 0$ by construction and $\kappa_t \in (0, 1)$ by choice. Therefore, we can deduce that (3.11c) will be satisfied for sufficiently small $\mu_k > 0$ by observing (3.30), (3.31) and (3.32). Combining this with (3.29) and the fact that the **while** loop on line 19 ensures that μ_k will eventually be as small as required, guarantees that the **while** loop will terminate finitely. This completes the proof as all remaining steps in the kth iteration are well-posed and explicit. □

3.3 Global convergence

We analyze the global convergence properties of Algorithm 5 under the assumption that the algorithm does not terminate finitely. That is, in this section we assume that neither a first-order optimal solution nor an infeasible stationary point is found after a finite number of iterations so that the sequence $\{(x_k, y_k, \mu_k)\}_{k\geq 0}$ is infinite.

We provide global convergence guarantees under the following assumption. This assumption is assumed throughout this section and is therefore not stated explicitly in each result.

Assumption 3.3.1. *The iterates $\{x_k\}_{k\geq 0}$ are contained in a convex compact set over which the objective function f and constraint function c are both twice-continuously differentiable.*

This assumption and the bound on the multipliers enforced in line 37 of Algorithm 5 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

$$\|\nabla^2_{xx}\mathcal{L}(\sigma, y_k, \mu_k)\|_2 \leq \Omega_j \text{ for all } \sigma \text{ on the segment } [x_k, x_k + s_k], \tag{3.33a}$$

$$\|\mu_k \nabla^2_{xx}\ell(x_k, y_k) + J_k^T J_k\|_2 \leq \Omega_j, \tag{3.33b}$$

$$\text{and } \|J_k^T J_k\|_2 \leq \Omega_j. \tag{3.33c}$$

We begin our analysis in this section by proving the following lemma, which provides critical bounds on differences in (components of) the AL function summed over sequences of iterations.

Lemma 3.3.1. *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

$$\sum_{k=0}^{p-1} \mu_k (f_k - f_{k+1}) < M_f, \tag{3.34}$$

$$\sum_{k=0}^{p-1} \mu_k y_k^T (c_{k+1} - c_k) < M_c, \tag{3.35}$$

$$\text{and } \sum_{k=0}^{p-1} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) < M_{\mathcal{L}}. \tag{3.36}$$

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

$$\sum_{k=0}^{\infty} \mu_k (f_k - f_{k+1}), \tag{3.37}$$

$$\sum_{k=0}^{\infty} \mu_k y_k^T (c_{k+1} - c_k), \tag{3.38}$$

$$\text{and} \quad \sum_{k=0}^{\infty} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) \tag{3.39}$$

converge and are finite, and

$$\lim_{k \to \infty} \|c_k\|_2 = \bar{c} \ \textit{ for some } \bar{c} \geq 0. \tag{3.40}$$

Proof. Under Assumption 3.3.1 we may conclude that for some constant $\overline{M}_f > 0$ and all integers $p \geq 1$ we have

$$\sum_{k=0}^{p-1} (f_k - f_{k+1}) = f_0 - f_p < \overline{M}_f.$$

If $\mu_k = \mu$ for all sufficiently large k, then this implies that (3.34) clearly holds for some sufficiently large M_f. Otherwise, if $\mu_k \to 0$, then it follows from Dirichlet's Test [51, §3.4.10] and the fact that $\{\mu_k\}_{k \geq 0}$ is a monotonically decreasing sequence that converges to zero that (3.37) converges and is finite.

Next, we show that for some constant $\overline{M}_c > 0$ and all integers $p \geq 1$ we have

$$\sum_{k=0}^{p-1} y_k^T (c_{k+1} - c_k) < \overline{M}_c. \tag{3.41}$$

First, suppose that $\mathcal{Y}$ defined in (3.16) is finite. It follows that there exists $k' \geq 0$ and y such that $y_k = y$ for all $k \geq k'$. Moreover, under Assumption 3.3.1 there exists a constant $\widehat{M}_c > 0$ such that for all $p \geq k' + 1$ we have

$$\sum_{k=k'}^{p-1} y_k^T (c_{k+1} - c_k) = y^T \sum_{k=k'}^{p-1} (c_{k+1} - c_k) = y^T (c_p - c_{k'}) \leq \|y\|_2 \|c_p - c_{k'}\|_2 < \widehat{M}_c.$$

It is now clear that (3.41) holds in this case. Second, suppose that $|\mathcal{Y}| = \infty$ so that the sequence $\{k_j\}_{j \geq 1}$ in Algorithm 5 is infinite. By construction $t_j \to 0$, so for some $j' \geq 1$ we have

$$t_j = t_{j-1}^{1+\epsilon} \text{ and } Y_{j+1} = t_{j-1}^{-\epsilon} \text{ for all } j \geq j'. \tag{3.42}$$

From the definition of the sequence $\{k_j\}_{j \geq 1}$, (3.14), and (3.15), we know that

$$\begin{aligned} \sum_{k=k_j}^{k_{j+1}-1} y_k^T (c_{k+1} - c_k) = y_{k_j}^T \sum_{k=k_j}^{k_{j+1}-1} (c_{k+1} - c_k) &= y_{k_j}^T (c_{k_{j+1}} - c_{k_j}) \\ &\leq \|y_{k_j}\|_2 \|c_{k_{j+1}} - c_{k_j}\|_2 \leq 2 Y_{j+1} t_j = 2 t_{j-1} \text{ for all } j \geq j', \end{aligned}$$

where the last equality follows from (3.42). Using these relationships, summing over all $j' \leq j \leq j' + q$ for an arbitrary integer $q \geq 1$, and using the fact that $t_{j+1} \leq \gamma_t t_j$

by construction, leads to

$$\begin{aligned}\sum_{j=j'}^{j'+q}\left(\sum_{k=k_j}^{k_{j+1}-1} y_k^T(c_{k+1}-c_k)\right) &\leq 2\sum_{j=j'}^{j'+q} t_{j-1} \\ &\leq 2t_{j'-1}\sum_{l=0}^{q}\gamma_t^l \\ &= 2t_{j'-1}\frac{1-\gamma_t^{q+1}}{1-\gamma_t} \\ &\leq \frac{2t_{j'-1}}{1-\gamma_t}.\end{aligned}$$

It is now clear that (3.41) holds in this case as well.

We have shown that (3.41) always holds. Thus, if $\mu_k = \mu$ for all sufficiently large k, then (3.35) holds for some sufficiently large M_c. Otherwise, if $\mu_k \to 0$, then it follows from Dirichlet's Test [51, §3.4.10], (3.41) and the fact that $\{\mu_k\}_{k\geq 0}$ is a monotonically decreasing sequence that converges to zero that (3.38) converges and is finite.

Finally, observe that

$$\begin{aligned}&\sum_{k=0}^{p-1}\mathcal{L}(x_k,y_k,\mu_k)-\mathcal{L}(x_{k+1},y_k,\mu_k) \\ =&\sum_{k=0}^{p-1}\mu_k(f_k-f_{k+1})+\sum_{k=0}^{p-1}\mu_k y_k^T(c_{k+1}-c_k)+\tfrac{1}{2}\sum_{k=0}^{p-1}(\|c_k\|_2^2-\|c_{k+1}\|_2^2) \\ =&\sum_{k=0}^{p-1}\mu_k(f_k-f_{k+1})+\sum_{k=0}^{p-1}\mu_k y_k^T(c_{k+1}-c_k)+\tfrac{1}{2}(\|c_0\|_2^2-\|c_p\|_2^2). \qquad (3.43)\end{aligned}$$

If $\mu_k = \mu$ for all sufficiently large k, then it follows from Assumption 3.3.1, (3.34), (3.35), and (3.43) that (3.36) will hold for some sufficiently large $M_{\mathcal{L}}$. Otherwise,

consider when $\mu_k \to 0$. Taking the limit of (3.43) as $p \to \infty$, we have from Assumption 3.3.1 and conditions (3.37) and (3.38) that

$$\sum_{k=0}^{\infty}(\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) < \infty.$$

Since the terms in this sum are all nonnegative, it follows from the Monotone Convergence Theorem that (3.39) converges and is finite. We may again take the limit of (3.43) as $p \to \infty$ and use (3.37), (3.38), and (3.39) to conclude that (3.40) holds. □

In the following subsections, we consider different situations depending on the number of times that the Lagrange multiplier vector is updated.

3.3.1 Finite number of multiplier updates

In this section, we consider the case when $\mathcal{Y}$ in (3.16) is finite. In this case, the counter j in Algorithm 5, which tracks the number of times that the dual vector is updated, satisfies

$$j \in \{1, 2, \ldots \bar{j}\} \text{ for some finite } \bar{j}. \tag{3.44}$$

For the purposes of our analysis in this section, we define

$$t := t_{\bar{j}} > 0 \text{ and } T := T_{\bar{j}} > 0. \tag{3.45}$$

We consider two subcases depending on whether the penalty parameter stays

bounded away from zero, or if it converges to zero. First we consider cases when it converges to zero.

Lemma 3.3.2. *If $|\mathcal{Y}| < \infty$ and $\mu_k \to 0$, then there exist a vector y and integer $\overline{k} \geq 0$ such that*

$$y_k = y \ \text{ for all } \ k \geq \overline{k}, \tag{3.46}$$

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

$$\lim_{k\to\infty} \|c_k\|_2 = \bar{c} > 0 \ \text{ and } \ \lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{3.47}$$

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

Proof. Since $|\mathcal{Y}| < \infty$, we know that (3.44) and (3.45) both hold for some $\bar{j} \geq 0$. It follows by construction in Algorithm 5 that there exists y and a scalar $\overline{k} \geq k_{\bar{j}}$ such that (3.46) holds.

From (3.40), it follows that $\|c_k\|_2 \to \bar{c}$ for some $\bar{c} \geq 0$. If $\bar{c} = 0$, then by Assumption 3.3.1, the definition of $\nabla_x \mathcal{L}$, (3.46), and the fact that $\mu_k \to 0$ it follows that $\lim_{\mu\to 0} \nabla_x \mathcal{L}(x_k, y, \mu_k) = J_k^T c_k = 0$, which implies that $\lim_{\mu\to 0} F_{\text{AL}}(x_k, y, \mu_k) = F_{\text{FEAS}}(x_k) = 0$. This would imply that for some $k \geq \overline{k}$ the algorithm would set $j \leftarrow \bar{j} + 1$, violating (3.44). Thus, we conclude that $\bar{c} > 0$, which proves the first part of (3.47).

Next, we prove that

$$\liminf_{k\geq 0} F_{\text{FEAS}}(x_k) = 0. \tag{3.48}$$

If (3.48) does not hold, then there exists $\zeta \in (0,1)$ and $k' \geq \overline{k}$ such that

$$\|F_{\text{FEAS}}(x_k)\|_2 \geq 2\zeta \;\text{ for all }\; k \geq k'. \tag{3.49}$$

Hence, by (3.49) and the fact that $\mu_k \to 0$, there exists $k'' \geq k'$ such that

$$\|F_{\text{AL}}(x_k, y, \mu_k)\|_2 \geq \zeta \;\text{ for all }\; k \geq k''. \tag{3.50}$$

We now show that the trust-region radius Θ_k is bounded away from zero for $k \geq k''$. In order to see this, suppose that for some $k \geq k''$ we have

$$0 < \Theta_k \leq \min\left\{ \frac{\zeta}{1+\Omega_{\bar{j}}}, \frac{(1-\eta_{vs})\kappa_1\kappa_5\zeta}{1+\Omega_{\bar{j}}} \right\} =: \delta_{\text{thresh}} \tag{3.51}$$

where $\{\Omega_j\}_{j\geq 1}$ is defined with (3.33). It then follows from (3.11a), (3.28), (3.33b), (3.50), and (3.51) that

$$\Delta q(s_k; x_k, y, \mu_k) \geq \kappa_1 \Delta q(\overline{s}_k; x_k, y, \mu_k) \geq \kappa_1\kappa_5\zeta \min\left\{ \frac{\zeta}{1+\Omega_{\bar{j}}}, \Theta_k \right\} \geq \kappa_1\kappa_5\zeta\Theta_k. \tag{3.52}$$

Using the definition of q, Taylor's Theorem, (3.33a), (3.33b), and the trust-region

constraint, we may conclude that for some σ_k on the segment $[x_k, x_k + s_k]$ we have

$$\begin{aligned} &|q(s_k; x_k, y, \mu_k) - \mathcal{L}(x_k + s_k, y, \mu_k)| \\ =&\tfrac{1}{2} \left| s_k^T(\mu_k \nabla^2_{xx} \ell(x_k, y_k) + J_k^T J_k) s_k - s_k^T \nabla^2_{xx} \mathcal{L}(\sigma_k, y, \mu_k) s_k \right| \\ \leq&\Omega_{\bar{j}} \|s_k\|_2^2 \leq \Omega_{\bar{j}} \Theta_k^2. \end{aligned} \tag{3.53}$$

The definition of ρ_k, (3.51), (3.52), and (3.53) then yield

$$|\rho_k - 1| = \left| \frac{q(s_k; x_k, y, \mu_k) - \mathcal{L}(x_k + s_k, y, \mu_k)}{\Delta q(s_k; x_k, y, \mu_k)} \right| \leq \frac{\Omega_{\bar{j}} \Theta_k^2}{\kappa_1 \kappa_5 \zeta \Theta_k} = \frac{\Omega_{\bar{j}} \Theta_k}{\kappa_1 \kappa_5 \zeta} \leq 1 - \eta_{vs}.$$

This implies that the **if** clause in line 26 of Algorithm 5 will be true, and along with (3.50) we may conclude that the trust-region radius will not be decreased any further. Consequently, we have shown that the trust-region radius updating strategy in Algorithm 5 guarantees that for some $\delta_{\min} \in (0, \delta_{\text{thresh}})$ we have

$$\Theta_k \geq \delta_{\min} \text{ for all } k \geq k''. \tag{3.54}$$

Now, since Θ_k is bounded below, there must exist an infinite subsequence, indexed by an ordered set $\mathcal{S} \subseteq \mathbb{N}$, of successful iterates. If we define $\mathcal{S}'' := \{k \in \mathcal{S} : k \geq k''\}$, then we may conclude from the fact that $x_{k+1} = x_k$ when $k \notin \mathcal{S}$, (3.46), (3.12), (3.52),

and (3.54) that

$$\begin{aligned}\sum_{k=k''}^{\infty} \mathcal{L}(x_k, y, \mu_k) - \mathcal{L}(x_{k+1}, y, \mu_k) &= \sum_{k \in \mathcal{S}''} \mathcal{L}(x_k, y, \mu_k) - \mathcal{L}(x_{k+1}, y, \mu_k) \\ &\geq \sum_{k \in \mathcal{S}''} \eta_s \Delta q(s_k; x_k, y, \mu_k) \\ &\geq \sum_{k \in \mathcal{S}''} \eta_s \kappa_1 \kappa_5 \zeta \delta_{\min} = \infty, \end{aligned} \tag{3.55}$$

contradicting (3.39). Therefore, we conclude that (3.48) holds.

Now we prove the second part of (3.47). By contradiction, suppose $F_{\text{FEAS}}(x_k) \not\to 0$. This supposition and (3.48) imply that the subsequence of successful iterates indexed by $\mathcal{S}$ is infinite and there exists a constant $\varepsilon \in (0, 1)$ and sequences $\{\underline{k}_i\}_{i \geq 0}$ and $\{\overline{k}_i\}_{i \geq 0}$ defined in the following manner: $\underline{k}_0$ is the first iterate in $\mathcal{S}$ such that $\|F_{\text{FEAS}}(x_{\underline{k}_0})\|_2 \geq 4\varepsilon > 0$; $\overline{k}_i$ for $i \geq 0$ is the first iterate strictly greater than $\underline{k}_i$ such that

$$\|F_{\text{FEAS}}(x_{\overline{k}_i})\|_2 < 2\varepsilon; \tag{3.56}$$

and $\underline{k}_i$ for $i \geq 1$ is the first iterate in $\mathcal{S}$ strictly greater than $\overline{k}_{i-1}$ such that

$$\|F_{\text{FEAS}}(x_{\underline{k}_i})\|_2 \geq 4\varepsilon > 0. \tag{3.57}$$

We may now define

$$\mathcal{K} := \{k \in \mathcal{S} : \underline{k}_i \leq k < \overline{k}_i \text{ for some } i \geq 0\}.$$

Since $\mu_k \to 0$, we may use (3.46) to conclude that there exists k''' such that

$$\|F_{\text{AL}}(x_k, y, \mu_k)\|_2 \geq \varepsilon \text{ for all } k \in \mathcal{K} \text{ such that } k \geq k'''. \tag{3.58}$$

It follows from the definition of $\mathcal{K}$, (3.11a), (3.28), (3.46), (3.33b), and (3.58) that

$$\begin{aligned}\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k) &\geq \eta_s \Delta q(s_k; x_k, y_k, \mu_k) \\ &\geq \eta_s \kappa_1 \kappa_5 \varepsilon \min\left\{\frac{\varepsilon}{1+\Omega_{\bar{j}}}, \Theta_k\right\}\end{aligned} \tag{3.59}$$

for all $k \in \mathcal{K}$ such that $k \geq k'''$. It also follows from (3.39) and since $\mathcal{L}(x_{k+1}, y_k, \mu_k) \leq \mathcal{L}(x_k, y_k, \mu_k)$ for all $k \geq 0$ that

$$\begin{aligned}\infty &> \sum_{k=0}^{\infty} \mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k) \\ &= \sum_{k \in \mathcal{S}} \mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k) \\ &\geq \sum_{k \in \mathcal{K}} \mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k).\end{aligned} \tag{3.60}$$

Summing (3.59) for $k \in \mathcal{K}$ and using (3.60) yields $\lim_{k \in \mathcal{K}} \Theta_k = 0$, which combined

with (3.59) and (3.46) leads to

$$\mathcal{L}(x_k, y, \mu_k) - \mathcal{L}(x_{k+1}, y, \mu_k) \geq \eta_s \kappa_1 \kappa_5 \varepsilon \Theta_k > 0 \tag{3.61}$$

for all sufficiently large $k \in \mathcal{K}$.

By the triangle-inequality and (3.61), there exists some $\bar{i} \geq 1$ such that

$$\begin{aligned}
\|x_{\underline{k}_i} - x_{\overline{k}_i}\|_2 &\leq \sum_{j=\underline{k}_i}^{\overline{k}_i - 1} \|x_j - x_{j+1}\|_2 \\
&= \sum_{j=\underline{k}_i}^{\overline{k}_i - 1} \|s_j\|_2 \\
&\leq \sum_{j=\underline{k}_i}^{\overline{k}_i - 1} \Theta_j \\
&\leq \frac{1}{\eta_s \kappa_1 \kappa_5 \varepsilon} \sum_{j=\underline{k}_i}^{\overline{k}_i - 1} \mathcal{L}(x_j, y, \mu_j) - \mathcal{L}(x_{j+1}, y, \mu_j) \;\text{ for }\; i \geq \bar{i}.
\end{aligned}$$

Summing over all $i \geq \bar{i}$ and using (3.39), we find

$$\begin{aligned}
\sum_{i=\bar{i}}^{\infty} \|x_{\underline{k}_i} - x_{\overline{k}_i}\|_2 &\leq \frac{1}{\eta_s \kappa_1 \kappa_5 \varepsilon} \sum_{i=\bar{i}}^{\infty} \left(\sum_{j=\underline{k}_i}^{\overline{k}_i - 1} \mathcal{L}(x_j, y, \mu_j) - \mathcal{L}(x_{j+1}, y, \mu_j) \right) \\
&\leq \frac{1}{\eta_s \kappa_1 \kappa_5 \varepsilon} \sum_{k \in \mathcal{K}} \mathcal{L}(x_k, y, \mu_k) - \mathcal{L}(x_{k+1}, y, \mu_k) \\
&\leq \frac{1}{\eta_s \kappa_1 \kappa_5 \varepsilon} \sum_{k=0}^{\infty} \mathcal{L}(x_k, y, \mu_k) - \mathcal{L}(x_{k+1}, y, \mu_k) < \infty,
\end{aligned}$$

which implies that

$$\lim_{i\to\infty} \|x_{\underline{k}_i} - x_{\overline{k}_i}\|_2 = 0.$$

It follows from the previous limit, (3.57), and Assumption 3.3.1 that for i sufficiently large we have $\|F_{\text{FEAS}}(x_{\overline{k}_i})\|_2 > 2\varepsilon$, contradicting (3.56). We may conclude that the second part of (3.47) holds.

The fact that every limit point of $\{x_k\}_{k\geq 0}$ is an infeasible stationary point follows from (3.47). □

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 augmented Lagrangian.

Lemma 3.3.3. *If $|\mathcal{Y}| < \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then with t defined in* (3.45) *there exist a vector y and integer $\overline{k} \geq 0$ such that*

$$y_k = y \;\; \textit{and} \;\; \|c_k\|_2 \geq t \;\; \textit{for all} \;\; k \geq \overline{k}, \tag{3.62}$$

and we have the limit

$$\lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{3.63}$$

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

Proof. Since $|\mathcal{Y}| < \infty$, we know that (3.44) and (3.45) both hold for some $\bar{j} \geq 0$. Since we also suppose that $\mu_k = \mu > 0$ for all sufficiently large k, it follows by construction

in Algorithm 5 that there exists y and a scalar $k' \geq k_{\bar{j}}$ such that

$$\mu_k = \mu \text{ and } y_k = y \text{ for all } k \geq k'. \tag{3.64}$$

Next, we prove that

$$\liminf_{k \geq 0} \|F_{\text{AL}}(x_k, y, \mu)\|_2 = 0. \tag{3.65}$$

If (3.65) does not hold, then there exists $\zeta \in (0, 1)$ and $k'' \geq k'$ such that

$$\|F_{\text{AL}}(x_k, y, \mu)\|_2 \geq \zeta \text{ for all } k \geq k''. \tag{3.66}$$

We now show that the trust-region radius Θ_k is bounded away from zero for $k \geq k''$. In order to see this, suppose that for some $k \geq k''$ we have

$$0 < \Theta_k \leq \min \left\{ \frac{\zeta}{1 + \Omega_{\bar{j}}}, \frac{(1 - \eta_{vs})\kappa_1 \kappa_5 \zeta}{1 + \Omega_{\bar{j}}} \right\} =: \delta_{\text{thresh}} \tag{3.67}$$

where $\{\Omega_j\}_{j \geq 1}$ is defined with (3.33). It then follows from (3.11a), (3.28), (3.33b), (3.66), and (3.67) that

$$\Delta q(s_k; x_k, y, \mu) \geq \kappa_1 \Delta q(\overline{s}_k; x_k, y, \mu) \geq \kappa_1 \kappa_5 \zeta \min \left\{ \frac{\zeta}{1 + \Omega_{\bar{j}}}, \Theta_k \right\} \geq \kappa_1 \kappa_5 \zeta \Theta_k. \tag{3.68}$$

Using the definition of q, Taylor's Theorem, (3.33a), (3.33b), and the trust-region

constraint, we may conclude that for some σ_k on the segment $[x_k, x_k + s_k]$ we have

$$\begin{aligned} &|q(s_k; x_k, y, \mu_k) - \mathcal{L}(x_k + s_k, y, \mu_k)| \\ =& \tfrac{1}{2} \left| s_k^T (\mu_k \nabla^2_{xx} \ell(x_k, y_k) + J_k^T J_k) s_k - s_k^T \nabla^2_{xx} \mathcal{L}(\sigma_k, y, \mu_k) s_k \right| \\ \leq& \Omega_{\bar{j}} \|s_k\|_2^2 \leq \Omega_{\bar{j}} \Theta_k^2. \end{aligned} \tag{3.69}$$

The definition of ρ_k, (3.69), (3.68), and (3.67) then yield

$$\begin{aligned} |\rho_k - 1| &= \left| \frac{q(s_k; x_k, y, \mu_k) - \mathcal{L}(x_k + s_k, y, \mu_k)}{\Delta q(s_k; x_k, y, \mu_k)} \right| \\ &\leq \frac{\Omega_{\bar{j}} \Theta_k^2}{\kappa_1 \kappa_5 \zeta \Theta_k} = \frac{\Omega_{\bar{j}} \Theta_k}{\kappa_1 \kappa_5 \zeta} \leq 1 - \eta_{vs}. \end{aligned}$$

This implies that a very successful iteration will occur and along with the trust-region radius updating strategy in Algorithm 5, we may conclude that for some $\delta_{\min} \in (0, \delta_{\text{thresh}})$ we have

$$\Theta_k \geq \delta_{\min} \text{ for all } k \geq k''. \tag{3.70}$$

Now, since Θ_k is bounded below, there must exist an infinite subsequence, indexed by an ordered set $\mathcal{S} \subseteq \mathbb{N}$, of successful iterates. If we define $\mathcal{S}'' := \{k \in \mathcal{S} : k \geq k''\}$, then we may conclude from the fact that $x_{k+1} = x_k$ when $k \notin \mathcal{S}$, (3.64), (3.12), (3.68),

and (3.70) that

$$\begin{aligned}\sum_{k=k''}^{\infty} \mathcal{L}(x_k, y, \mu) - \mathcal{L}(x_{k+1}, y, \mu) &= \sum_{k \in \mathcal{S}''} \mathcal{L}(x_k, y, \mu) - \mathcal{L}(x_{k+1}, y, \mu) \\ &\geq \sum_{k \in \mathcal{S}''} \eta_s \Delta q(s_k; x_k, y, \mu) \\ &\geq \sum_{k \in \mathcal{S}''} \eta_s \kappa_1 \kappa_5 \zeta \delta_{\min} = \infty, \end{aligned} \tag{3.71}$$

contradicting (3.36). Therefore, we conclude that (3.65) holds. Moreover, the same argument used in the second part of the proof of Lemma 3.3.2 (with F_{FEAS} replaced by F_{AL}) shows that

$$\lim_{k \to \infty} \|F_{\text{AL}}(x_k, y, \mu)\|_2 = 0. \tag{3.72}$$

It then follows that there exists $\overline{k} \geq k'$ such that $\|c_k\|_2 \geq t$ for all $k \geq \overline{k}$, since otherwise it follows from (3.72) that for some $k \geq \overline{k}$ Algorithm 5 sets $j \leftarrow \bar{j} + 1$, violating (3.44). Thus, we have shown that (3.62) holds.

Next, we turn to the limits in (3.63). It follows from (3.72) and part (i) of Lemma 3.2.3 that

$$\lim_{k \to \infty} \|s_k\|_2 \leq \lim_{k \to \infty} \Theta_k = \lim_{k \to \infty} \Gamma_k \min\{\delta_k, \delta \|P[x_k - \nabla_x \mathcal{L}(x_k, y, \mu)] - x_k\|_2\} = 0. \tag{3.73}$$

From (3.73) and Assumption 3.3.1, we have

$$\lim_{k \to \infty} \Delta q_v(s_k; x_k) = 0, \tag{3.74}$$

and from the definition of v, (3.45), and (3.62) that

$$v_k - \tfrac{1}{2}(\kappa_t t_{\bar{j}})^2 \geq \tfrac{1}{2}t^2 - \tfrac{1}{2}(\kappa_t t)^2 = \tfrac{1}{2}(1-\kappa_t^2)t^2 > 0 \text{ for all } k \geq \overline{k}. \tag{3.75}$$

We now prove that $F_{\text{FEAS}}(x_k) \to 0$. To see this, first note that

$$F_{\text{AL}}(x_k, y, \mu) \neq 0 \text{ for all } k \geq \overline{k},$$

or else the algorithm would set $\mu_{k+1} < \mu$ in line 12 of Algorithm 5, violating (3.64). Standard trust-region theory[50] then ensures that there will be infinitely many successful iterations, which we denote by $\mathcal{S}$, for $k \geq \overline{k}$. If we suppose that $F_{\text{FEAS}}(x_k) \not\to 0$, then for some $\zeta \in (0,1)$ there exists an infinite subsequence indexed by

$$\mathcal{S}_\zeta := \{k \in \mathcal{S} : k \geq \overline{k} \text{ and } \|F_{\text{FEAS}}(x_{k+1})\|_2 \geq \zeta\}.$$

We may then observe from the updating strategies for θ_k and δ_k that

$$\begin{aligned}
\theta_{k+1} &= \min\{\delta_{k+1}, \delta\|F_{\text{FEAS}}(x_{k+1})\|_2\} \\
&\geq \min\{\max\{\delta_R, \delta_k\}, \delta\zeta\} &(3.76)\\
&\geq \min\{\delta_R, \delta\zeta\} > 0 \text{ for all } k \in \mathcal{S}_\zeta. &(3.77)
\end{aligned}$$

Using (3.11b), (3.27), (3.33c), (3.44), and (3.77), we then find for $k \in \mathcal{S}_\zeta$ that

$$\begin{aligned} \Delta q_v(r_{k+1}; x_{k+1}) &\geq \kappa_2 \Delta q_v(\overline{r}_{k+1}; x_{k+1}) \\ &\geq \kappa_2 \kappa_4 \zeta \min\left\{ \frac{\zeta}{1+\Omega_{\bar{j}}}, \delta_R, \delta\zeta \right\} =: \zeta' > 0. \end{aligned} \tag{3.78}$$

We may now combine (3.78), (3.75), and (3.74) to state that (3.11c) must be violated for sufficiently large $k \in \mathcal{S}_\zeta$ and, consequently, the penalty parameter will be decreased. However, this is a contradiction to (3.64), so we conclude that $F_{\text{FEAS}}(x_k) \to 0$. The fact that every limit point of $\{x_k\}_{k\geq 0}$ is an infeasible stationary point follows since $\|c_k\|_2 \geq t$ for all $k \geq \overline{k}$ from (3.62) and $F_{\text{FEAS}}(x_k) \to 0$. □

This completes the analysis for the case that the set $\mathcal{Y}$ is finite. The next section considers the complementarity situation when the Lagrange multiplier vector is updated an infinite number of times.

3.3.2 Infinite number of multiplier updates

We now consider the case when $|\mathcal{Y}| = \infty$. In this case, it follows by construction in Algorithm 5 that

$$\lim_{j\to\infty} t_j = \lim_{j\to\infty} T_j = 0. \tag{3.79}$$

As in the previous subsection, we consider two subcases depending on whether the penalty parameter remains bounded away from zero, or if it converges to zero. Our

next lemma shows that when the penalty parameter does remain bounded away from zero, then a subsequence of iterates converges to a first-order optimal point. In general, this is the ideal case for a feasible problem.

Lemma 3.3.4. *If $|\mathcal{Y}| = \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then it follows that*

$$\lim_{j\to\infty} c_{k_j} = 0 \tag{3.80}$$

$$\text{and} \quad \lim_{j\to\infty} F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j}) = 0. \tag{3.81}$$

Thus, any limit point (x_*, y_*) of $\{(x_{k_j}, \widehat{y}_{k_j})\}_{j\geq 0}$ is first-order optimal for (2.1).

Proof. Since $|\mathcal{Y}| = \infty$, the condition in line 32 holds an infinite number of times. The limit (3.80) then follows by (3.79) since line 35 sets $k_j \leftarrow k + 1$ for all $k_j \in \mathcal{Y}$.

To prove (3.81), we first define

$$\mathcal{Y}' = \{k_j \in \mathcal{Y} : \|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2 \leq \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2\}.$$

It follows from (3.79) and line 34 of Algorithm 5 that if $\mathcal{Y}'$ is infinite, then

$$\lim_{k_j\in\mathcal{Y}'} F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j}) = 0. \tag{3.82}$$

Meanwhile, it follows from (3.79) and line 34 of Algorithm 5 that if the set $\mathcal{Y}\backslash\mathcal{Y}'$ is

infinite, then

$$\lim_{k_j \in \mathcal{Y} \setminus \mathcal{Y}'} F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}) = 0. \tag{3.83}$$

Under Assumption 3.3.1, (3.83) may be combined with (3.80) and the fact that $\mu_k = \mu$ for some $\mu > 0$ to deduce that if $\mathcal{Y} \setminus \mathcal{Y}'$ is infinite, then

$$\lim_{k_j \in \mathcal{Y} \setminus \mathcal{Y}'} F_{\text{L}}(x_{k_j}, y_{k_j-1}) = 0. \tag{3.84}$$

We may now combine (3.84) with (3.13) to state that if $\mathcal{Y} \setminus \mathcal{Y}'$ is infinite, then

$$\lim_{k_j \in \mathcal{Y} \setminus \mathcal{Y}'} F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j}) = 0. \tag{3.85}$$

The desired result (3.81) now follows from (3.82), (3.85), and the supposition that $|\mathcal{Y}| = \infty$. This completes the proof. □

We now prove a corollary showing that if Algorithm 5 employs a particular update for $\widehat{y}_{k+1}$ in line 33 (satisfying (3.13)), then a subsequence of multiplier estimates $\{\widehat{y}_k\}$ converges to an optimal Lagrange multiplier vector when the linear independence constraint qualification (LICQ) holds at limit points. For this result only, we make the following additional assumption.

Assumption 3.3.2. *If x_* is a limit point of $\{x_k\}$ that is feasible for problem (2.1), then $\mathcal{I}(x_*) := \{i : [x_*]_i > 0\} \neq \emptyset$ and the matrix $J_{\mathcal{I}}(x_*)$ that contains the subset of columns of $J(x_*)$ corresponding to the index set $\mathcal{I}(x_*)$ has full row rank.*

Corollary 3.3.1. *If $|\mathcal{Y}| = \infty$, $\mu_k = \mu > 0$ for all sufficiently large k, and line 33 of Algorithm 5 sets*

$$\widehat{y}_{k+1} \leftarrow \begin{cases} y_k & \text{if } \|F_{L}(x_{k+1}, y_k)\|_2 \leq \|F_{L}(x_{k+1}, \pi(x_{k+1}, y_k, \mu_k))\|_2, \\ \pi(x_{k+1}, y_k, \mu_k) & \text{otherwise,} \end{cases} \tag{3.86}$$

then there exists an infinite ordered set $\mathcal{J} \subseteq \mathbb{N}$ such that

$$\lim_{j \in \mathcal{J}} (x_{k_j}, y_{k_j}) = (x_*, y_*),$$

where (x_*, y_*) a first-order optimal point for problem (2.1), the vector y_* is the (unique) solution of

$$\min_{y \in \mathbb{R}^m} \ \|g_{\mathcal{I}}(x_*) - J_{\mathcal{I}}(x_*)^T y\|_2^2, \tag{3.87}$$

and $g_{\mathcal{I}}(x_*)$ and $\mathcal{J}_{\mathcal{I}}(x_*)^T$ contain the rows of $g(x_*)$ and $\mathcal{J}(x_*)^T$, respectively, corresponding to $\mathcal{I}(x_*)$.

Proof. We know from $|\mathcal{Y}| = \infty$ and Assumption 3.3.1 that there exists an infinite ordered set $\mathcal{J}_1 \subseteq \mathbb{N}$ and a vector x_* such that $\lim_{j \in \mathcal{J}_1} x_{k_j} = x_*$. It also follows from this fact, the assumptions of this corollary, and Lemma 3.3.4 that

$$\lim_{j \in \mathcal{J}_1} c(x_{k_j}) = 0, \quad \lim_{j \in \mathcal{J}_1} F_{\mathrm{L}}(x_{k_j}, \widehat{y}_{k_j}) = 0, \tag{3.88}$$

and any limit points of $(x_{k_j}, \widehat{y}_{k_j})$ are first-order KKT points for (2.1). Let us define the set

$$\mathcal{J}_2 := \{j \in \mathcal{J}_1 : \|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2 \leq \|F_{\text{AL}}(x_{k_j}, \pi(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}))\|_2\},$$

which is motivated by line 34 of Algorithm 5. Also, we note that

$$\lim_{j \in \mathcal{J}_1} Y_j = \infty \tag{3.89}$$

as a result of (3.79) and line 35 of Algorithm 5. We now consider two cases.

Case 1: Suppose that $|\mathcal{J}_2| = \infty$. It follows from (3.79), line 34 of Algorithm 5, and definition of $\mathcal{J}_2$ that

$$0 = \lim_{j \in \mathcal{J}_2} \|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2 = \lim_{j \in \mathcal{J}_2} \|P\big[x_{k_j} - \big(g(x_{k_j}) - J(x_{k_j})^T \widehat{y}_{k_j}\big)\big] - x_{k_j}\|_2.$$

Using this limit and the definition of $\mathcal{I}(x_*)$, it follows that

$$\lim_{j \in \mathcal{J}_2} \big(g_{\mathcal{I}}(x_{k_j}) - J_{\mathcal{I}}(x_{k_j})^T \widehat{y}_{k_j}\big) = 0,$$

which, combined with $\lim_{j \in \mathcal{J}_1} x_{k_j} = x_*$, Assumption 3.3.2, and (3.87), implies that

$$\lim_{j \in \mathcal{J}_2} \widehat{y}_{k_j} = y_* \tag{3.90}$$

so that $(x_*, y_*) = \lim_{j \in \mathcal{J}_2}(x_{k_j}, \widehat{y}_{k_j})$ is a first-order point for (2.1). Using (3.90), the fact that the upper bound on the multipliers increases to infinity in (3.89), and the definition of y_{k+1} in line 37, we then have

$$\lim_{j \in \mathcal{J}_2} y_{k_j} = y_*.$$

Defining $\mathcal{J} := \mathcal{J}_2 \subseteq \mathcal{J}_1$, this completes the proof for this case.

Case 2: Suppose that $|\mathcal{J}_2| < \infty$. Since $|\mathcal{J}_2| < \infty$, it follows from (3.79), line 34 of Algorithm 5, and the fact that $\mu_k = \mu > 0$ for all sufficiently large k that

$$\begin{aligned} 0 &= \lim_{j \in \mathcal{J}_1} \|F_{\text{AL}}(x_{k_j}, \pi(x_{k_j}, y_{k_j-1}, \mu), \mu)\|_2 \\ &= \lim_{j \in \mathcal{J}_1} \|P\big[x_{k_j} - \mu\big(g(x_{k_j}) - J(x_{k_j})^T \pi(x_{k_j}, y_{k_j-1}, \mu)\big)\big] - x_{k_j}\|_2. \end{aligned}$$

Using this limit, the definition of $\mathcal{I}$, and $\lim_{j \in \mathcal{J}_1} x_{k_j} = x_*$ shows that

$$\lim_{j \in \mathcal{J}_1} g_{\mathcal{I}}(x_{k_j}) - J_{\mathcal{I}}(x_{k_j})^T \pi(x_{k_j}, y_{k_j-1}, \mu) = 0,$$

which, under Assumption 3.3.2, yields

$$\lim_{j \in \mathcal{J}_1} \pi(x_{k_j}, y_{k_j-1}, \mu) = y_*. \tag{3.91}$$

It then follows from this fact and (3.88) that

$$\lim_{j \in \mathcal{J}_1} y_{k_j - 1} = y_*. \tag{3.92}$$

Combining (3.91) and (3.92) with (3.86) implies that

$$\lim_{j \in \mathcal{J}_1} \widehat{y}_{k_j} = y_* \tag{3.93}$$

so that $(x_*, y_*) = \lim_{j \in \mathcal{J}_1} (x_{k_j}, \widehat{y}_{k_j})$ is a first-order KKT point for problem (2.1). Finally, combining (3.93), the fact that the upper bound on the multipliers increases to infinity in (3.89), and the definition of y_{k+1} in line 37 of Algorithm 5, we have

$$\lim_{j \in \mathcal{J}_1} y_{k_j} = y_*.$$

Defining $\mathcal{J} := \mathcal{J}_1$, this completes the proof for this case. □

Finally, we consider the case when the penalty parameter converges to zero. For this case, we require the following technical lemma.

Lemma 3.3.5. *Suppose $l \leq x \leq u$ and let v be any vector in $\mathbb{R}^n$. Then, for any scalar $\xi > 1$ we have*

$$\|P[x + \xi v] - x\|_2 \leq \xi \|P[x + v] - x\|_2.$$

Proof. It suffices to prove the result for the case when $l \leq 0 \leq u$ and $x = 0$ since the proof for the more general case is similar. We may write

$$\begin{aligned} P[\xi v] &= P[v] + (P[\xi v] - P[v]) &=: P[v] + w_1 \\ \text{and } \xi P[v] &= P[v] + (\xi - 1)P[v] &=: P[v] + w_2, \end{aligned}$$

where, since $\xi > 1$, we have for all $i \in \{1, \ldots, n\}$ that

$$[w_1]_i = \begin{cases} 0 & \text{if } v_i \leq l_i \\ 0 & \text{if } v_i \geq u_i \\ l_i - v_i & \text{if } l_i < v_i < u_i \text{ and } \xi v_i \leq l_i \\ u_i - v_i & \text{if } l_i < v_i < u_i \text{ and } \xi v_i \geq u_i \\ (\xi - 1)v_i & \text{if } l_i < \xi v_i < u_i \end{cases} \quad \text{and}$$

$$[w_2]_i = \begin{cases} (\xi - 1)l_i & \text{if } v_i \leq l_i \\ (\xi - 1)u_i & \text{if } v_i \geq u_i \\ (\xi - 1)v_i & \text{if } l_i < v_i < u_i \text{ and } \xi v_i \leq l_i \\ (\xi - 1)v_i & \text{if } l_i < v_i < u_i \text{ and } \xi v_i \geq u_i \\ (\xi - 1)v_i & \text{if } l_i < \xi v_i < u_i. \end{cases}$$

Hence, it is easily verified that

$$P[v]^T w_1 \leq P[v]^T w_2 \quad \text{and} \quad \|w_1\|_2^2 \leq \|w_2\|_2^2,$$

which along with the identity $\frac{1}{2}\|P[v]+w\|_2^2 = \frac{1}{2}\|P[v]\|_2^2 + P[v]^T w + \frac{1}{2}\|w\|_2^2$ yields the desired result. □

We now prove the following lemma, which reveals that if there are an infinite number of multiplier updates and the penalty parameter converges to zero, then the constraint violation measure converges to zero. Moreover, in such cases, as long as the number of decreases of the penalty parameter between consecutive multiplier updates is bounded, then any limit point of one of two possible subsequences is a first-order optimal point for (2.1).

Lemma 3.3.6. *If $|\mathcal{Y}| = \infty$ and $\mu_k \to 0$, then*

$$\lim_{k\to\infty} c_k = 0. \tag{3.94}$$

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

$$\lim_{j\in\mathcal{J}, j\to\infty} \|F_{L}(x_{k_j}, \widehat{y}_{k_j})\|_2 = 0 \quad \text{or} \quad \lim_{j\in\mathcal{J}, j\to\infty} \|F_{L}(x_{k_j}, \pi(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}))\|_2 = 0. \tag{3.95}$$

In such cases, if the first (respectively, second) limit in (3.95) *holds, then along with*

(3.94) *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 optimal point for problem* (2.1)*.*

Proof. It follows from (3.40) that

$$\lim_{k\to\infty} \|c_k\|_2 = \bar{c} \geq 0. \tag{3.96}$$

However, it also follows from (3.79) and line 32 of Algorithm 5 that

$$\lim_{j\to\infty} \|c_{k_j}\|_2 \leq \lim_{j\to\infty} t_j = 0. \tag{3.97}$$

The limit (3.94) now follows from (3.96) and (3.97).

To prove the remainder of the result, first note that by lines 34 and 36 of Algorithm 5 and since

$$0 = \lim_{k\to\infty} \mu_k = \lim_{j\to\infty} \mu_{k_j}, \tag{3.98}$$

we have for all sufficiently large j that

$$\min\{\|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2, \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2\} \leq T_j = \gamma_T \mu_{k_{j-1}-1} T_{j-1}. \tag{3.99}$$

If there exists an infinite ordered set $\mathcal{J} \subseteq \mathbb{N}$ such that $\lim_{j\in\mathcal{J}, j\to\infty} \|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2 = 0$, then the first limit in (3.95) holds and there is nothing left to prove. Thus, suppose that $\{\|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2\}$ is bounded below and away from zero for all sufficiently large

j. Then, from (3.98) and (3.99), we have

$$\|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2 \leq \gamma_T \mu_{k_{j-1}-1} T_{j-1}$$

for all sufficiently large j, from which it follows along with Lemma 3.3.5 that

$$\begin{aligned}
\gamma_T T_{j-1} &\geq \tfrac{1}{\mu_{k_{j-1}-1}} \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2 \\
&\geq \|P[x_{k_j} - \tfrac{1}{\mu_{k_{j-1}-1}} \nabla_x \mathcal{L}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})] - x_{k_j}\|_2 \\
&= \|P[x_{k_j} - \tfrac{\mu_{k_j-1}}{\mu_{k_{j-1}-1}} (g(x_{k_j}) - J(x_{k_j})^T \pi(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}))] - x_{k_j}\|_2.
\end{aligned}$$

With these inequalities, (3.79), and the fact that $\mu_{k_j-1}/\mu_{k_{j-1}-1} \in [\gamma_\mu^p, 1]$ for all sufficiently large j, Lemma 3.2.1 (taking a further infinite subset of $\mathcal{J}$, if necessary) yields the second limit in (3.95). □

3.3.3 Overall global convergence result

We combine the lemmas in the previous subsections to obtain the following result.

Theorem 3.3.1. *One of the following must hold true:*

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

(ii) $\mu_k \not\to 0$ 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 optimal for (2.1); 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 optimal for problem* (2.1).

Proof. Lemmas 3.3.2, 3.3.3, 3.3.4, and 3.3.6 cover the only four possible outcomes of Algorithm 5; the result follows from those described in these lemmas. □

We complete this chapter with a discussion of Theorem 3.3.1. As with all penalty methods for nonconvex optimization, Algorithm 5 may converge to a local minimizer of the constraint violation that is infeasible, i.e., an infeasible stationary point (see Definition 2.1.2). This possible outcome is, in fact, unavoidable since we have not (implicitly) assumed that problem (2.1) is feasible. By far, the most common outcome of our numerical results in Section 5.1 is case (ii) of Theorem 3.3.1, in which the penalty parameter ultimately remains fixed and convergence to a first-order primal-dual solution of (2.1) is observed. In fact, these numerical tests show that our adaptive algorithm is far more efficient than our basic implementation and, importantly, at least as reliable. The final possible outcome is that the penalty parameter and the constraint violation both converge to zero. In this case we are unable to guarantee that limit points are first-order solutions of (2.1), even under the assumption that the MFCQ holds, and therefore have obtained a weaker convergence result than many other augmented Lagrangian methods. Nonetheless, we remain content since the numerical tests in Section 5.1 show that Algorithm 5 is superior to the basic approach and that the penalty parameter consistently remains bounded away from zero.

Of course, it is possible to view our adaptive strategy as a mechanism for quickly obtaining an improved Lagrange multiplier estimate and value for the penalty parameter. These values may then be used as the initial input for a traditional augmented Lagrangian method. For example, motivated by Lemma 3.3.6, one could employ our algorithm, but transition to a traditional augmented Lagrangian method once the Lagrange multiplier estimate has been updated more than a prescribed number of times, the quantities μ_k and $\|c(x_k)\|$ are below prescribed positive tolerances, and the penalty parameter has been decreased more than a prescribed number of times since the most recent Lagrange multiplier estimate update. This simple strategy inherits the well-documented convergence theory for standard augmented Lagrangian methods and benefits from the practical advantages of steering exhibited by our approach.

Finally, it is also possible to modify Algorithm 5 so that convergence to first-order optimal points may be established even in case (iii) of Theorem 3.3.1. Specifically, we could make the following changes: (i) compute first-order multiplier estimates $\widehat{y}_{k+1}$ during *every* iteration (whereas currently they are only computed when $\|c_{k+1}\|_2 \leq t_j$); (ii) switch the order of the two **if** statements in Lines 32 and 34 of Algorithm 5; (iii) explicitly limit the number of decreases of the penalty parameter allowed between updates to the multiplier vector (as motivated by part (iii) of Theorem 3.3.1); and (iv) if the explicit bound on the number of updates allowed by part (iii) is reached, then do not allow a further decrease to the penalty parameter until either the multiplier is updated again, or an approximate minimizer of the augmented Lagrangian is found at

which the constraint violation is not sufficiently small, i.e., not less than t_j. Although these changes could be made to Algorithm 5, we have chosen not to do so for two reasons. First, these additions would further complicate the algorithm in a manner that we do not believe is justified from a practical perspective. Second, again from a practical perspective, it may be inefficient to compute new multiplier estimates during every iteration.

Chapter 4

An Adaptive AL Line-Search Method

In this chapter, we present and analyze an adaptive AL line search method which is a variant of the AL trust-region method in Chapter 3. We show that the algorithm is well-posed and achieves the same global convergence results as the adaptive AL trust-region method.

4.1 Algorithm description

Our adaptive AL line search algorithm is similar to the adaptive AL trust-region method proposed in Chapter 3, except for two key differences: (i) it executes line searches rather than using a trust-region framework, and (ii) it employs a convexi-

fied piecewise quadratic model of the AL function for computing the search direction during 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 procedure. 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 §5.1.1. Similar to the trust-region method proposed in Chapter 3, 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 (2.1) (or at least (2.4)) 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:

$$q_v(s; x) = \tfrac{1}{2}\|c(x) + J(x)s\|_2^2$$

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

We note that q_v is the same as in (3.1), and $\tilde{q}$ is a convexification of the model q (see

(3.3)) that we used in our trust-region method. (We should point out that $\tilde{q}$ is not necessarily a convex function in s. It is convex along any direction and serves the purpose in our setting.)

Just like the trust-region method in Chapter 3, our linesearch 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

$$\underset{r\in\mathbb{R}^n}{\text{minimize}}\ q_v(r;x_k)\ \text{ subject to }\ l \le x_k + r \le u,\ \|r\|_2 \le \theta_k, \tag{4.1}$$

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

$$\theta_k := \delta\|F_{\text{FEAS}}(x_k)\|_2 \ge 0. \tag{4.2}$$

Problem (4.1) is the same as (3.4) except for the way we define the trust-region radius θ_k. Since later on we will use a backtracking line search strategy to update our primal variable (see (4.8)), we do not have the term δ_k here in the definition of θ_k. But similar to (3.5), this choice of trust region forces the steering step to be smaller in norm as the iterates of the algorithm approach any stationary point of the constraint violation measure.[48] This prevents the steering step from being too large relative to the progress that can be made toward minimizing v. Like the trust-region algorithm in the previous chapter, our algorithm only requires r_k to be an approximate solution of (4.1). 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 (4.7a) 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

$$\Delta q_v(\overline{r}_k; x_k) := q_v(0; x_k) - q_v(\overline{r}_k; x_k) \geq -\varepsilon_r \overline{r}_k^T J_k^T c_k \ \text{ and } \ \|\overline{r}_k\|_2 \leq \theta_k. \tag{4.3}$$

This definition of a Cauchy step is exactly the same as that in 3.6. Therefore, a Cauchy step and 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—can be computed from Algorithm 3. 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 4.2.4. We define a similar reduction $\Delta q_v(r_k; x_k)$ for the steering step r_k.

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

$$\underset{s \in \mathbb{R}^n}{\text{minimize}} \ \tilde{q}(s; x_k, y_k, \mu_k) \ \text{ subject to } \ l \leq x_k + s \leq u, \ \|s\|_2 \leq \Theta_k, \tag{4.4}$$

where, given $\Gamma_k > 1$ from the output of Algorithm 3, we define the trust-region radius

$$\Theta_k := \Theta(x_k, y_k, \mu_k, \Gamma_k) = \Gamma_k \delta \|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2 \geq 0. \tag{4.5}$$

Problem (4.4) is different from (3.8) in two ways. First, we minimize the $\tilde{q}$ model instead of the q model. This will allow our line search strategy to work (see Lemma 4.2.6).

Second, the trust region is defined in a slightly different way since our line search algorithm does not require δ_k any more, but rather a positive constant δ. Similar to the steering step, we allow inexactness in the solution of (4.4) by only requiring the step s_k to satisfy a Cauchy decrease condition (see (4.7a)), where the Cauchy step for problem (4.4) is

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

for $\overline{\alpha}_k = \overline{\alpha}(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$ such that, for $\varepsilon_k \geq 0$ returned from Algorithm 3, the vector $\overline{s}_k$ yields

$$\begin{aligned} \Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) &:= \tilde{q}(0; x_k, y_k, \mu_k) - \tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) \\ &\geq -\frac{(\varepsilon_k + \varepsilon_r)}{2}\overline{s}_k^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k) \ \text{ and } \ \|\overline{s}_k\|_2 \leq \Theta_k. \end{aligned} \tag{4.6}$$

Algorithm 6 describes our procedure for computing $\overline{\alpha}_k$ and $\overline{s}_k$. (The importance of incorporating Γ_k in (4.5) and ε_k in (4.6) is revealed in the proofs of Lemmas 4.2.2 and 4.2.3. Algorithm 6 is very similar to Algorithm 4 with the only difference being line 5 where we use the convexified model. The quantity $\Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k)$ representing the predicted reduction in $\mathcal{L}(\cdot, y_k, \mu_k)$ yielded by $\overline{s}_k$ 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 4.2.4. A similar quantity $\Delta\tilde{q}(s_k; x_k, y_k, \mu_k)$ is also used for the direction s_k.

Algorithm 6 Cauchy step computation for the AL subproblem (4.4)

1: **procedure** CAUCHY_AL_LS($x_k, y_k, \mu_k, \Theta_k, \varepsilon_k$)
2: **restrictions** : $\mu_k > 0$, $\Theta_k \geq 0$, and $\varepsilon_k \geq 0$.
3: **available constants** : $\{\varepsilon_r, \gamma\} \subset (0, 1)$.
4: Set $\overline{\alpha}_k \leftarrow 1$ and $\overline{s}_k \leftarrow P[x_k - \overline{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)] - x_k$.
5: **while** (4.6) is not satisfied **do**
6: Set $\overline{\alpha}_k \leftarrow \gamma \overline{\alpha}_k$ and $\overline{s}_k \leftarrow P[x_k - \overline{\alpha}_k \nabla_x \mathcal{L}(x_k, y_k, \mu_k)] - x_k$.
7: **end while**
8: **return** : $(\overline{\alpha}_k, \overline{s}_k)$
9: **end procedure**

Our complete algorithm is given as Algorithm 7. In particular, the kth iteration proceeds as follows. Given the kth iterate tuple (x_k, y_k, μ_k), the algorithm first determines whether a first-order solution or an infeasible stationary point to problem (2.1) is found. (Recall Definition 2.1.1 and Definition 2.1.2.) If either is the case, then the algorithm terminates, but otherwise the method enters the **while** loop in line 11 to check for stationarity with respect to the AL function. This loop is guaranteed to terminate finitely; see Lemma 4.2.1. Up to now, the kth iteration of Algorithm 7 is exactly the same as for Algorithm 5. 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 19, the overall goal of this block is to compute (approximate) solutions of subproblems (4.1) and (4.4) satisfying

$$\Delta\tilde{q}(s_k; x_k, y_k, \mu_k) \geq \kappa_1 \Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) > 0,\ l \leq x_k + s_k \leq u,\ \|s_k\|_2 \leq \Theta_k, \quad (4.7a)$$

$$\Delta q_v(r_k; x_k) \geq \kappa_2 \Delta q_v(\overline{r}_k; x_k) \geq 0,\ l \leq x_k + r_k \leq u,\ \|r_k\|_2 \leq \theta_k, \quad (4.7b)$$

$$\text{and}\ \ \Delta q_v(s_k; x_k) \geq \min\{\kappa_3 \Delta q_v(r_k; x_k), v_k - \tfrac{1}{2}(\kappa_t t_j)^2\}. \quad (4.7c)$$

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 (4.1) and (4.4) satisfy (4.7), but for our purposes (see Theorem 4.2.1) it is sufficient that, for sufficiently small $\mu > 0$, they are at least satisfied by $r_k = \overline{r}_k$ and $s_k = \overline{s}_k$. The conditions in (4.7) are similar to the conditions in (3.11), so the motivations underlying (4.7) can be found in §3.1. In short, (3.11a) and (3.11b) are Cauchy decrease conditions while (3.11c) 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

$$\mathcal{L}(x_k + \gamma_\alpha^l s_k, y_k, \mu_k) \leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s \gamma_\alpha^l \Delta\widetilde{q}(s_k; x_k, y_k, \mu_k). \tag{4.8}$$

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 the same as that in Algorithm 5 and is composed of potential modifications of the Lagrange multiplier vector and target values t_j and T_j.

Algorithm 7 Adaptive Augmented Lagrangian Line Search Algorithm

Choose $\{\gamma, \gamma_\mu, \gamma_\alpha, \gamma_t, \gamma_T, \kappa_1, \kappa_2, \kappa_3, \varepsilon_r, \kappa_t\} \subset (0,1)$ and $\{\delta, \epsilon, Y\} \subset (0, \infty)$.
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).
  else 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 (4.2).
  Use Algorithm 3 to compute $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) \leftarrow$ CAUCHY_FEASIBILITY(x_k, θ_k).
  Set Θ_k by (4.5).
  Use Algorithm 6 to compute $(\overline{\alpha}_k, \overline{s}_k) \leftarrow$ CAUCHY_AL_LS$(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
  Obtain approx. solutions r_k / s_k to (4.1)/(4.4) satisfying (4.7a)–(4.7b).
  while (4.7c) 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 (4.5).
    Use Algorithm 6 to obtain $(\overline{\alpha}_k, \overline{s}_k) \leftarrow$ CAUCHY_AL_LS$(x_k, y_k, \mu_k, \Theta_k, \varepsilon_k)$.
    Compute an approximate solution s_k to (4.4) satisfying (4.7a).
  end while
  Set $\alpha_k \leftarrow \gamma_\alpha^l$ where $l \geq 0$ is the smallest integer satisfying (4.8).
  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 (3.13).
    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 $T_{j+1} \leftarrow \gamma_T \min\{1, \mu_k\} T_j$.
      Set y_{k+1} from (3.14) where α_y satisfies (3.15).
      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 $\mu_{k+1} \leftarrow \mu_k$ and then set $k \leftarrow k+1$.
end loop

4.2 Well-posedness

In this section, we show that Algorithm 7 is well posed. In order to show well-posedness of the algorithm, we make the same assumption as in Chapter 3.

Assumption 4.2.1. *At each given x_k, the objective function f and constraint function c are both twice-continuously differentiable.*

Our proof of the well-posedness of Algorithm 7 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 11 of Algorithm 7 terminates finitely. Since the same loop appears in Algorithm 5 and the proof of the result in that case is the same as that for Algorithm 7, we need only refer to Lemma 3.2.2 in order to state the following lemma for Algorithm 7.

Lemma 4.2.1. *If line 11 of Algorithm 7 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 7 are similar to those employed in Algorithm 5, we may state the following lemma showing that Algorithms 3 and 6 are well defined when called in lines 15, 17, and 21 of Algorithm 7. It should be noted that a slight difference between Algorithm 6 and Algorithm 4 is the use of the convexified model $\tilde{q}$ in (4.6). However, we claim that this difference does not affect the veracity of the result.

Lemma 4.2.2. *The following hold true:*

(i) The computation of $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k)$ *in line 15 of Algorithm 7 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 17 and 21 of Algorithm 7 is well defined.*

The next result, similar to Lemma 3.2.4, highlights critical relationships between q_v and $\tilde{q}$ as $\mu \to 0$. Indeed, much of the proof follows exactly the same logic as for Lemma 3.2.4, but we provide a complete proof to account for our present use of the convexified model $\tilde{q}$ and the differences in the trust-region radii for the subproblems employed in the algorithm. This result is crucial for showing that the steering condition (4.7c) is satisfied for all sufficient small μ (see Lemma 4.2.5).

Lemma 4.2.3. *Let* $(\overline{\beta}_k, \overline{r}_k, \varepsilon_k, \Gamma_k) \leftarrow$ CAUCHY_FEASIBILITY(x_k, θ_k) *with* θ_k *defined by* (4.2) *and, as quantities dependent on the penalty parameter* $\mu > 0$*, let us write* $(\overline{\alpha}_k(\mu), \overline{s}_k(\mu)) \leftarrow$ CAUCHY_AL_LS$(x_k, y_k, \mu, \Theta_k(\mu), \varepsilon_k)$ *where we also define the quantity* $\Theta_k(\mu) := \Gamma_k \delta \|F_{AL}(x_k, y_k, \mu)\|_2$ *(see* (4.5)*). Then, the following hold true:*

$$\lim_{\mu \to 0} \left(\max_{\|s\|_2 \leq 2\theta_k} |\tilde{q}(s; x_k, y_k, \mu) - q_v(s; x_k)| \right) = 0, \tag{4.9a}$$

$$\lim_{\mu \to 0} \nabla_x \mathcal{L}(x_k, y_k, \mu) = J_k^T c_k, \tag{4.9b}$$

$$\lim_{\mu \to 0} \overline{s}_k(\mu) = \overline{r}_k, \tag{4.9c}$$

$$\text{and} \quad \lim_{\mu \to 0} \Delta q_v(\overline{s}_k(\mu); x_k) = \Delta q_v(\overline{r}_k; x_k). \tag{4.9d}$$

Proof. Since x_k and y_k are fixed during iteration k, for ease of exposition we often

drop these quantities from function dependencies for the purposes of this proof. From the definitions of q_v and $\tilde{q}$, it follows that for some constants $M_1 > 0$ and $M_2 > 0$ independent of μ we have

$$\begin{aligned}\max_{\|s\|_2 \leq 2\theta_k} |\tilde{q}(s;\mu) - q_v(s)| &= \max_{\|s\|_2 \leq 2\theta_k} |\mu\ell_k + \mu\nabla_x \ell_k^T s + \max\{\tfrac{\mu}{2} s^T \nabla_{xx}\ell_k s, -\tfrac{1}{2}\|J_k s\|_2^2\}| \\ &\leq \mu M_1 + \max\{\mu M_2, -\tfrac{1}{2}\|J_k s\|_2^2\}.\end{aligned}$$

Since the right-hand side of this expression vanishes as $\mu \to 0$, we have (4.9a). Furthermore, it holds that

$$\nabla_x \mathcal{L}(x_k, y_k, \mu) - J_k^T c_k = \mu(g_k - J_k^T y_k),$$

which implies that (4.9b) holds.

We now show that (4.9c) holds. We consider two cases depending on the value $F_{\text{FEAS}}(x_k)$. Throughout, it should be observed that all quantities in Algorithm 3 are unaffected by μ, so they can be considered as fixed quantities.

Case 1: If $F_{\text{FEAS}}(x_k) = 0$, then $\theta_k = \delta\|F_{\text{FEAS}}(x_k)\|_2 = 0$, from which it follows that $\overline{r}_k = 0$ and $\Delta q_v(\overline{r}_k) = 0$. Furthermore, from (4.9b), we have $\Theta_k(\mu) \to 0$ as $\mu \to 0$, which means $\overline{s}_k(\mu) \to 0 = \overline{r}_k$, as desired.

Case 2: Now suppose that $F_{\text{FEAS}}(x_k) \neq 0$. In the following arguments, we define the

following functions of a nonnegative integer l and positive scalar μ:

$$r_k(l) = P[x_k - \gamma^l J_k^T c_k] - x_k \quad \text{and} \quad s_k(l,\mu) = P[x_k - \gamma^l \nabla_x \mathcal{L}(\mu)] - x_k.$$

We also define $l_\beta \geq 0$ to be the integer such that $\overline{\beta}_k = \gamma^{l_\beta}$ (see Algorithm 3), which implies that

$$\overline{r}_k = r_k(l_\beta). \tag{4.10}$$

We have as a consequence of (4.9b) that

$$\lim_{\mu \to 0} s_k(l,\mu) = r_k(l) \ \text{ for any } \ l \geq 0.$$

In particular, this implies with (4.10) that

$$\lim_{\mu \to 0} s_k(l_\beta,\mu) = r_k(l_\beta) = \overline{r}_k. \tag{4.11}$$

Thus, (4.9c) follows as long as

$$\overline{s}_k(\mu) = s_k(l_\beta,\mu) \quad \text{for all sufficiently small} \quad \mu > 0. \tag{4.12}$$

Since the computation of $\overline{s}_k(\mu)$ (via the CAUCHY_AL_LS routine stated as

Algorithm 6) computes a nonnegative integer $l_{\alpha,\mu}$ such that

$$\overline{s}_k(\mu) = P[x_k - \gamma^{l_{\alpha,\mu}} \nabla_x \mathcal{L}(\mu)] - x_k,$$

it follows that (4.12) can be proved by showing that $l_{\alpha,\mu} = l_\beta$ for all sufficiently small $\mu > 0$. As a preliminary result in the proof of this fact, we first show that for l_k computed in Algorithm 3 we have

$$\min\{l_\beta, l_{\alpha,\mu}\} \geq l_k \quad \text{for all sufficiently small} \quad \mu > 0. \tag{4.13}$$

Indeed, if $l_k = 0$, then (4.13) holds trivially. Thus, let us suppose that $l_k > 0$. According to the procedures in Algorithm 3, it is clear that $l_\beta \geq l_k$. Hence, we may turn our attention to $l_{\alpha,\mu}$. From the definition of $\Theta_k(\mu)$ (in the statement of this lemma), (4.9b), the manner in which Γ_k is set in Algorithm 3, the fact that $\theta_k > 0$, and since $\|P[x_k - \gamma^{l_k - 1} J_k^T c_k] - x_k\|_2 > \theta_k$ due to the manner in which l_k is set in Algorithm 3, we have that

$$\begin{aligned}
\lim_{\mu \to 0} \Theta_k(\mu) &= \lim_{\mu \to 0} \Gamma_k \delta \|F_{\text{AL}}(x_k, y_k, \mu)\|_2 = \Gamma_k \delta \|F_{\text{FEAS}}(x_k)\|_2 = \Gamma_k \theta_k \\
&= \min\left\{2, \tfrac{1}{2}\left(1 + \frac{\|P[x_k - \gamma^{l_k - 1} J_k^T c_k] - x_k\|_2}{\theta_k}\right)\right\} \theta_k \\
&= \min\left\{2\theta_k, \tfrac{1}{2}\left(\theta_k + \|P[x_k - \gamma^{l_k - 1} J_k^T c_k] - x_k\|_2\right)\right\} \\
&\in \left(\theta_k, \|P[x_k - \gamma^{l_k - 1} J_k^T c_k] - x_k\|_2\right).
\end{aligned}$$

Along with (4.9b), this implies that for all sufficiently small $\mu > 0$ we have

$$\lim_{\mu \to 0} \|P[x_k - \gamma^{l_k - 1} \nabla_x \mathcal{L}(\mu)] - x_k\|_2 = \|P[x_k - \gamma^{l_k - 1} J_k^T c_k] - x_k\|_2 > \Theta_k(\mu).$$

This shows that $l_{\alpha,\mu} \geq l_k$ holds for all sufficiently small $\mu > 0$. Consequently, we have (4.13).

Having ensured that (4.13) holds, we proceed to prove that $l_{\alpha,\mu} = l_\beta$ for all sufficiently small $\mu > 0$. It follows from the definition of l_β, (4.10), the procedures of Algorithm 3 (e.g., the manner in which ε_k is set), and part (i) of Lemma 4.2.2 that

$$\begin{aligned} &-\frac{\Delta q_v(\overline{r}_k)}{\overline{r}_k^T J_k^T c_k} = -\frac{\Delta q_v\big(r_k(l_\beta)\big)}{r_k(l_\beta)^T J_k^T c_k} \geq \epsilon_r \\ &\text{and} \quad -\frac{\Delta q_v\big(r_k(l)\big)}{r_k(l)^T J_k^T c_k} \leq \epsilon_k < \epsilon_r \ \text{ for all integers } l_k \leq l < l_\beta. \end{aligned} \tag{4.14}$$

(Here, it is important to note that [50, Theorem 12.1.4] can be invoked to ensure that all denominators in (4.14) are negative.) It follows from (4.9b), (4.11), (4.9a), (4.14), and part (i) of Lemma 4.2.2 that

$$\lim_{\mu \to 0} -\frac{\Delta \tilde{q}\big(s_k(l_\beta, \mu)\big)}{s_k(l_\beta, \mu)^T \nabla_x \mathcal{L}(\mu)} = -\frac{\Delta q_v(\overline{r}_k)}{\overline{r}_k^T J_k^T c_k} \geq \varepsilon_r > \frac{\varepsilon_k + \varepsilon_r}{2} \tag{4.15}$$

and

$$\begin{aligned}\lim_{\mu\to 0} -\frac{\Delta\tilde{q}\big(s_k(l,\mu)\big)}{s_k(l,\mu)^T\nabla_x\mathcal{L}(\mu)} &= -\frac{\Delta q_v\big(r_k(l)\big)}{r_k(l)^T J_k^T c_k} \\ &\leq \varepsilon_k < \frac{\varepsilon_k+\varepsilon_r}{2} \quad \text{for all integers} \quad l_k \leq l < l_\beta.\end{aligned} \tag{4.16}$$

It now follows from (4.13), (4.15), (4.16), and (4.6) that $l_{\alpha,\mu} = l_\beta$ for all sufficiently small $\mu > 0$. As previously mentioned, this proves (4.9c).

Finally, note that (4.9d) follows from (4.9c) and continuity of q_v. □

We also need the following lemma related to Cauchy decreases in the models q_v and $\tilde{q}$. The conclusions of the lemma are similar to Lemma 3.2.5, but here we account for the convexified model $\tilde{q}$ and other differences in the subproblems employed here as opposed to those in Chapter 3.

Lemma 4.2.4. *Let Ω be any scalar value such that*

$$\Omega \geq \max\{\|\mu_k\nabla^2_{xx}\ell(x_k,y_k) + J_k^TJ_k\|_2, \|J_k^TJ_k\|_2\}. \tag{4.17}$$

Then, the following hold true:

(i) For some $\kappa_4 \in (0,1)$, the Cauchy step for subproblem (4.1) yields

$$\Delta q_v(\overline{r}_k; x_k) \geq \kappa_4\|F_{FEAS}(x_k)\|_2^2 \min\left\{\delta, \frac{1}{1+\Omega}\right\}. \tag{4.18}$$

(ii) For some $\kappa_5 \in (0,1)$, the Cauchy step for subproblem (4.4) yields

$$\Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) \geq \kappa_5 \|F_{AL}(x_k, y_k, \mu_k)\|_2^2 \min\left\{\delta, \frac{1}{1+\Omega}\right\}. \tag{4.19}$$

Proof. Let $\Sigma_k := 1 + \sup\{|\omega_k(r)| : 0 < \|r\|_2 \leq \theta_k\}$, where

$$\omega_k(r) = \frac{-\Delta q_v(r; x_k) - r^T J_k^T c_k}{\|r\|_2^2} \text{ for all } r \in \mathbb{R}^n.$$

In fact, using (4.17), the Cauchy-Schwartz inequality, and standard norm inequalities, we have that

$$\omega_k(r) = \frac{r^T J_k^T J_k r}{2\|r\|_2^2} \leq \Omega \text{ for all } r \in \mathbb{R}^n.$$

Hence, $\Sigma_k \leq 1 + \Omega$. The requirement (4.3) and [49, Theorem 4.4] then yield, for some $\bar{\kappa}_4 \in (0,1)$, that

$$\Delta q_v(\overline{r}_k; x_k) \geq \epsilon_r \bar{\kappa}_4 \|F_{\text{FEAS}}(x_k)\|_2 \min\left\{\theta_k, \frac{1}{\Sigma_k}\|F_{\text{FEAS}}(x_k)\|_2\right\}.$$

which, with (4.2), implies that (4.18) follows with $\kappa_4 := \epsilon_r \bar{\kappa}_4$.

We now show (4.19) in a similar manner. To this end, let $\bar{\Sigma}_k := 1 + \sup\{|\bar{\omega}_k(s)| : 0 < \|s\|_2 \leq \Theta_k\}$ where

$$\bar{\omega}_k(s) := \frac{-\Delta\tilde{q}(s; x_k, y_k, \mu_k) - s^T \nabla_x \mathcal{L}(x_k, y_k, \mu_k)}{\|s\|_2^2} \text{ for all } s \in \mathbb{R}^n.$$

Using (4.17), we have in a similar manner as above that

$$\begin{aligned}\bar{\omega}_k(s) &= \frac{\max\{\mu_k s^T \nabla^2_{xx}\ell(x_k, y_k, \mu_k)s + s^T J_k^T J_k s, 0\}}{2\|s\|_2^2} \\ &\leq \frac{\|\mu_k \nabla^2_{xx}\ell(x_k, y_k, \mu_k) + J_k^T J_k\|_2 \|s\|_2^2}{2\|s\|_2^2} \leq \Omega.\end{aligned}$$

Thus, $\bar{\Sigma}_k \leq 1 + \Omega$. The requirement (4.6) and [49, Theorem 4.4] then yield, for some $\bar{\kappa}_5 \in (0, 1)$, that

$$\Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) \geq \frac{\epsilon_k + \epsilon_r}{2}\bar{\kappa}_5 \|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2 \min\left\{\Theta_k, \frac{1}{\bar{\Sigma}_k}\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2\right\},$$

which, with (4.5) and Lemma 4.2.2(i) yields (4.19) with $\kappa_5 := \frac{1}{2}\epsilon_r\bar{\kappa}_5$. □

The next lemma shows that the **while** loop at line 19 of Algorithm 7, which is responsible for ensuring that our adaptive steering conditions in (4.7) are satisfied, terminates finitely. The proof of this is similar to the second part of Theorem 3.2.1.

Lemma 4.2.5. *The **while** loop at line 19 of Algorithm 7 terminates finitely.*

Proof. Since Lemma 4.2.1 ensures that the latter condition in the **while** loop is satisfied for all sufficiently small $\mu_k > 0$, it suffices to show that $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$ satisfy (4.7c) for all sufficiently small $\mu_k > 0$. To see this, we may borrow notation from Lemma 4.2.3—i.e., to consider $s_k = \overline{s}_k$ as a quantity dependent on a parameter

$\mu > 0$—and observe that (4.9d) implies

$$\lim_{\mu \to 0} \Delta q_v(s_k(\mu); x_k) = \lim_{\mu \to 0} \Delta q_v(\overline{s}_k(\mu); x_k) = \Delta q_v(\overline{r}_k; x_k) = \Delta q_v(r_k; x_k). \tag{4.20}$$

If $\Delta q_v(r_k; x_k) > 0$, then (4.20) implies that (4.7c) is satisfied for sufficiently small $\mu_k > 0$. Otherwise,

$$\Delta q_v(r_k; x_k) = \Delta q_v(\overline{r}_k; x_k) = 0, \tag{4.21}$$

which along with (4.18) implies that $F_{\text{FEAS}}(x_k) = 0$. We may now consider two cases depending on whether x_k is feasible for (2.1). If $c_k \neq 0$, then Algorithm 7 would have terminated in line 9, meaning that the **while** loop at line 19 would not have been reached. On the other hand, if $c_k = 0$, then (4.21) implies

$$\min\{\kappa_3 \Delta q_v(r_k; x_k), v_k - \tfrac{1}{2}(\kappa_t t_j)^2\} = -\tfrac{1}{2}(\kappa_t t_j)^2 < 0. \tag{4.22}$$

This last strict inequality follows since $t_j > 0$ by construction and $\kappa_t \in (0, 1)$ by choice. Therefore, we can deduce that (4.7c) will be satisfied for sufficiently small $\mu_k > 0$ by observing (4.20), (4.21) and (4.22). □

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 4.2.6. *At line 24 of Algorithm 7, the search direction s_k is a strict descent*

direction for $\mathcal{L}(\cdot, y_k, \mu_k)$ *from* x_k*. In particular,*

$$\nabla_x \mathcal{L}(x_k, y_k, \mu_k)^T s_k \leq -\Delta\tilde{q}(s_k; x_k, y_k, \mu_k) \leq -\kappa_1 \Delta\tilde{q}(\overline{s}_k; x_k, y_k, \mu_k) < 0. \tag{4.23}$$

Proof. From the definition of $\tilde{q}$, we find

$$\begin{aligned}
\Delta\tilde{q}(s_k; x_k, y_k, \mu_k) &= \tilde{q}(0; x_k, y_k, \mu_k) - \tilde{q}(s_k; x_k, y_k, \mu_k) \\
&= -\nabla_x \mathcal{L}(x_k, y_k, \mu_k)^T s_k - \max\{\tfrac{1}{2} s_k^T (\mu_k \nabla^2_{xx} \ell(x_k, y_k) + J_k^T J_k) s_k, 0\} \\
&\leq -\nabla_x \mathcal{L}(x_k, y_k, \mu_k)^T s_k.
\end{aligned}$$

It follows from this inequality and (4.7a) that

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

as desired. □

We are now ready to present the theorem for the well-posedness of Algorithm 7.

Theorem 4.2.1. *Suppose that Assumption 4.2.1 holds. Then the kth iteration of Algorithm 7 is well posed. That is, either the algorithm will terminate in line 7 or 9, or it will compute* $\mu_k > 0$ *such that* $F_{AL}(x_k, y_k, \mu_k) \neq 0$ *and for the steps* $s_k = \overline{s}_k$ *and* $r_k = \overline{r}_k$ *the conditions in* (4.7) *will be satisfied, in which case* $(x_{k+1}, y_{k+1}, \mu_{k+1})$ *will be computed.*

Proof. If, during the kth iteration, Algorithm 7 terminates in line 7 or 9, then there is nothing to prove. Thus, to proceed in the proof, we may assume that line 11 is reached. Lemma 4.2.1 then ensures that

$$F_{\text{AL}}(x_k, y_k, \mu) \neq 0 \text{ for all sufficiently small } \mu > 0. \tag{4.24}$$

Consequently, the **while** loop in line 11 will terminate for a sufficiently small $\mu_k > 0$. Next, by construction, conditions (4.7a) and (4.7b) are satisfied for any $\mu_k > 0$ by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. Lemma 4.2.5 then shows that for a sufficiently small $\mu_k > 0$, (4.7c) is also satisfied by $s_k = \overline{s}_k$ and $r_k = \overline{r}_k$. Therefore, line 24 will be reached. Finally, Lemma 4.2.6 ensures that α_k in line 24 is well-defined. This completes the proof as all remaining lines in the kth iteration are explicit. □

4.3 Global convergence

According to Theorem 4.2.1, we have that Algorithm 7 will either terminate finitely or produce an infinite sequence of iterates. If it terminates finitely—which can only occur if line 7 or 9 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. Thus, it remains to focus on the global convergence properties of Algorithm 7 when the sequence $\{(x_k, y_k, \mu_k)\}$ is infinite. For such cases, we make the following additional assumption as in Chapter 3.

Assumption 4.3.1. *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.*

We shall tacitly presume that Assumption 4.3.1 holds throughout this section, and not state it explicitly. This assumption and the bound on the multipliers enforced in line 31 of Algorithm 7 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

$$\|\nabla^2_{xx}\mathcal{L}(\sigma, y_k, \mu_k)\|_2 \leq \Omega_j \text{ for all } \sigma \text{ on the segment } [x_k, x_k + s_k], \tag{4.25a}$$

$$\|\mu_k \nabla^2_{xx}\ell(x_k, y_k) + J_k^T J_k\|_2 \leq \Omega_j, \tag{4.25b}$$

$$\text{and } \|J_k^T J_k\|_2 \leq \Omega_j. \tag{4.25c}$$

In the subsequent analysis, we make use of the subset of iterations for which line 29 of Algorithm 7 is reached. To this end, we define the iteration index set, as in Chapter 3, to be

$$\mathcal{Y} := \left\{k_j : \|c_{k_j}\|_2 \leq t_j,\ \min\{\|F_{\text{L}}(x_{k_j}, \widehat{y}_{k_j})\|_2, \|F_{\text{AL}}(x_{k_j}, y_{k_j-1}, \mu_{k_j-1})\|_2\} \leq T_j\right\}. \tag{4.26}$$

We begin our analysis by giving the following result which provides critical bounds on differences in (components of) the AL summed over sequences of iterations. The proof is the same as in Lemma 3.3.1 and essentially relies on Assumption 4.3.1 and

Dirichlet's Test [51, §3.4.10].

Lemma 4.3.1. *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

$$\sum_{k=0}^{p-1} \mu_k (f_k - f_{k+1}) < M_f, \tag{4.27}$$

$$\sum_{k=0}^{p-1} \mu_k y_k^T (c_{k+1} - c_k) < M_c, \tag{4.28}$$

$$\text{and} \quad \sum_{k=0}^{p-1} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) < M_{\mathcal{L}}. \tag{4.29}$$

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

$$\sum_{k=0}^{\infty} \mu_k (f_k - f_{k+1}), \tag{4.30}$$

$$\sum_{k=0}^{\infty} \mu_k y_k^T (c_{k+1} - c_k), \tag{4.31}$$

$$\text{and} \quad \sum_{k=0}^{\infty} (\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)) \tag{4.32}$$

converge and are finite, and

$$\lim_{k \to \infty} \|c_k\|_2 = \bar{c} \ \text{ for some } \bar{c} \geq 0. \tag{4.33}$$

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

Lemma 4.3.2. *There exists a positive monotonically decreasing sequence* $\{C_j\}_{j\geq 1}$ *such that, with the sequence* $\{k_j\}$ *computed in Algorithm 7, the step-size sequence* $\{\alpha_k\}$ *satisfies*

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

Proof. By Taylor's Theorem and Lemma 4.2.6, it follows under Assumption 4.3.1 that there exists $\tau > 0$ such that for all sufficiently small $\alpha > 0$ we have

$$\mathcal{L}(x_k + \alpha s_k, y_k, \mu_k) - \mathcal{L}(x_k, y_k, \mu_k) \leq -\alpha \Delta\tilde{q}(s_k; x_k, y_k, \mu_k) + \tau\alpha^2\|s_k\|^2. \tag{4.34}$$

On the other hand, during the line search implicit in line 24 of Algorithm 7, a step-size α is rejected if

$$\mathcal{L}(x_k + \alpha s_k, y_k, \mu_k) - \mathcal{L}(x_k, y_k, \mu_k) > -\eta_s \alpha \Delta\tilde{q}(s_k; x_k, y_k, \mu_k). \tag{4.35}$$

Combining (4.34), (4.35), and (4.7a) 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 $\gamma_\alpha \in (0,1)$, (4.7a), (4.19), (4.25b), (4.5), and $\Gamma_k \in (1,2]$ (see Lemma 4.2.2) 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.

4.3.1 Finite number of multiplier updates

In this subsection, we suppose that the set $\mathcal{Y}$ in (4.26) is finite in that the counter j in Algorithm 7 satisfies

$$j \in \{1, 2, \ldots \bar{j}\} \text{ for some finite } \bar{j}. \tag{4.36}$$

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

$$t := t_{\bar{j}} > 0 \text{ and } T := T_{\bar{j}} > 0. \tag{4.37}$$

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 (2.1) is infeasible. Indeed, in this case, we show that every limit point of the primal iterate sequence is an infeasible stationary point as given by Definition 2.1.2.

Lemma 4.3.3. *If $|\mathcal{Y}| < \infty$ and $\mu_k \to 0$, then there exist a vector y and integer $\overline{k} \geq 0$ such that*

$$y_k = y \ \text{ for all } \ k \geq \overline{k}, \tag{4.38}$$

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

$$\lim_{k\to\infty} \|c_k\|_2 = \bar{c} > 0 \ \text{ and } \ \lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{4.39}$$

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

Proof. It follows from (4.36), (4.37), and the manner in which the multiplier estimates are updated in Algorithm 7 that there exists y and a scalar $\overline{k} \geq k_{\bar{j}}$ such that (4.38) holds. Thus, all that remains is to prove that (4.39) holds for some $\bar{c} > 0$. From (4.33) and the supposition that $\mu_k \to 0$, it follows that $\|c_k - 2\|_2 \to \bar{c}$ for some $\bar{c} \geq 0$. If $\bar{c} = 0$, then by Assumption 4.3.1, (4.38), and the fact that $\mu_k \to 0$ it follows that $\lim_{k\to\infty} \nabla_x \mathcal{L}(x_k, y, \mu_k) = \lim_{k\to\infty} J_k^T c_k = 0$, which implies that $\lim_{k\to\infty} F_{\text{AL}}(x_k, y, \mu_k) = \lim_{k\to\infty} F_{\text{FEAS}}(x_k) = 0$. This would imply that for some $k \geq \overline{k}$

the algorithm would set $j \leftarrow \bar{j} + 1$, thus violating (4.36). Consequently, we may conclude that $\bar{c} > 0$, which proves the first limit in (4.39). Now, to reach a contradiction to the second limit in (4.39), suppose that $F_{\text{FEAS}}(x_k) \nrightarrow 0$. This, together with Assumption 4.3.1, (4.38), and the supposition that $\mu_k \to 0$, implies that there exist a positive constant ϵ and an infinite index set $\mathcal{K}$ such that

$$\|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2 \geq \epsilon \text{ for all } k \in \mathcal{K}. \tag{4.40}$$

It follows from (4.8), (4.7a), (4.19), (4.40), and Lemma 4.3.2 we know that the following holds for all $k \in \mathcal{K}$:

$$\begin{aligned}
\mathcal{L}(x_{k+1}, y_k, \mu_k) &= \mathcal{L}(x_k + \alpha_k s_k, y_k, \mu_k) \\
&\leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s \alpha_k \Delta\tilde{q}(s_k; x_k, y_k, \mu_k) \\
&\leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s \alpha_k \kappa_1 \Delta\tilde{q}(\bar{s}_k; x_k, y_k, \mu_k) \\
&\leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s \alpha_k \kappa_1 \kappa_5 \|F_{\text{AL}}(x_k, y_k, \mu_k)\|_2^2 \min\left\{\delta, \frac{1}{1 + \Omega_{\bar{j}}}\right\} \\
&\leq \mathcal{L}(x_k, y_k, \mu_k) - \eta_s C_{\bar{j}} \kappa_1 \kappa_5 \epsilon^2 \min\left\{\delta, \frac{1}{1 + \Omega_{\bar{j}}}\right\}.
\end{aligned}$$

This implies that, for all $k \in \mathcal{K}$, the reduction $\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k)$ is greater than or equal to a positive constant. In the meantime, we know from Lemma 4.2.6 and the way we update x_k at each iteration that $\mathcal{L}(x_k, y_k, \mu_k) - \mathcal{L}(x_{k+1}, y_k, \mu_k) \geq 0$ for all k. Therefore, we have reached a contradiction to (4.32). This implies that our

supposition that $F_{\text{FEAS}}(x_k) \nrightarrow 0$ cannot be true, so we have (4.39). □

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 4.3.4. *If $|\mathcal{Y}| < \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then with t defined in (4.37) there exist a vector y and integer $\overline{k} \geq 0$ such that*

$$y_k = y \;\; \textit{and} \;\; \|c_k\|_2 \geq t \;\; \textit{for all} \;\; k \geq \overline{k}, \tag{4.41}$$

and we have the limit

$$\lim_{k\to\infty} F_{FEAS}(x_k) = 0. \tag{4.42}$$

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

Proof. Since $|\mathcal{Y}| < \infty$, we know that (4.36) and (4.37) hold for some $\bar{j} \geq 0$, and since we suppose that $\mu_k = \mu > 0$ for all sufficiently large k, it follows by the mechanisms for updating the Lagrange multiplier estimates in Algorithm 7 that there exists y and a scalar $k' \geq k_{\bar{j}}$ such that

$$\mu_k = \mu \;\text{ and }\; y_k = y \;\text{ for all }\; k \geq k'. \tag{4.43}$$

Our next goal is to prove that

$$\lim_{k\to\infty} \|F_{\text{AL}}(x_k, y, \mu)\|_2 = 0. \tag{4.44}$$

Indeed, to reach a contradiction, suppose that (4.44) does not hold. It then follows that there exist a positive number ζ and an infinite index set $\mathcal{K}'$ with all elements greater than or equal to k' such that

$$\|F_{\text{AL}}(x_k, y, \mu)\|_2 \geq \zeta \text{ for all } k \in \mathcal{K}'. \tag{4.45}$$

Similar to the proof of Lemma 4.3.3, it then follows from (4.45), (4.8), (4.7a), (4.19), and Lemma 4.3.2 that for all $k \in \mathcal{K}'$ we have

$$\begin{aligned}
\mathcal{L}(x_{k+1}, y, \mu) &= \mathcal{L}(x_k + \alpha_k s_k, y, \mu) \\
&\leq \mathcal{L}(x_k, y, \mu) - \eta_s \alpha_k \Delta\tilde{q}(s_k; x_k, y, \mu) \\
&\leq \mathcal{L}(x_k, y, \mu) - \eta_s \alpha_k \kappa_1 \Delta\tilde{q}(\overline{s}_k; x_k, y, \mu) \\
&\leq \mathcal{L}(x_k, y, \mu) - \eta_s \alpha_k \kappa_1 \kappa_5 \|F_{\text{AL}}(x_k, y, \mu)\|_2^2 \min\left\{\delta, \frac{1}{1+\Omega_{\bar{j}}}\right\} \\
&\leq \mathcal{L}(x_k, y, \mu) - \eta_s C_{\bar{j}} \kappa_1 \kappa_5 \zeta^2 \min\left\{\delta, \frac{1}{1+\Omega_{\bar{j}}}\right\}.
\end{aligned}$$

This implies that, for all $k \in \mathcal{K}'$, the reduction $\mathcal{L}(x_k, y, \mu) - \mathcal{L}(x_{k+1}, y, \mu)$ is greater than or equal to a positive constant. However, Assumption 4.3.1 implies that $\mathcal{L}(x_k, y, \mu)$ is bounded below. Therefore, we have reached a contradiction, so (4.44) must hold.

The first consequence of (4.44) is that it allows us to prove (4.41). Indeed, it follows that there exists $\overline{k} \geq k'$ such that $\|c_k\|_2 \geq t$ for all $k \geq \overline{k}$, since otherwise (4.44) would imply that, for some $k \geq \overline{k}$, Algorithm 7 would set $j \leftarrow \bar{j} + 1$, which violates (4.36). Thus, along with (4.43), we have proved (4.41).

The second consequence of (4.44) is that it allows us to prove (4.42), which is all that remains to complete the proof of the lemma. It follows from (4.7a), (4.44), and part (i) of Lemma 4.2.2 that

$$\lim_{k\to\infty} \|s_k\|_2 \leq \lim_{k\to\infty} \Theta_k = \lim_{k\to\infty} \Gamma_k \delta \|F_{\text{AL}}(x_k, y, \mu)\|_2 = 0. \tag{4.46}$$

Furthermore, from (4.46) and Assumption 4.3.1, we have

$$\lim_{k\to\infty} \Delta q_v(s_k; x_k) = 0, \tag{4.47}$$

and, along with (4.37) and (4.41), we have

$$v_k - \tfrac{1}{2}(\kappa_t t_{\bar{j}})^2 \geq \tfrac{1}{2}t^2 - \tfrac{1}{2}(\kappa_t t)^2 = \tfrac{1}{2}(1 - \kappa_t^2)t^2 > 0 \ \text{ for all } \ k \geq \overline{k}. \tag{4.48}$$

We may use these facts to prove $F_{\text{FEAS}}(x_k) \to 0$. In particular, in order to derive a contradiction, suppose that $F_{\text{FEAS}}(x_k) \nrightarrow 0$. Then, there exist a positive number ξ

and an infinite index set $\mathcal{K}''$ such that

$$\|F_{\text{FEAS}}(x_k)\|_2 \geq \xi \text{ for all } k \in \mathcal{K}''. \tag{4.49}$$

Using (4.7b), (4.18), (4.25c), and (4.36), we then find for $k \in \mathcal{K}''$ that

$$\Delta q_v(r_k; x_k) \geq \kappa_2 \Delta q_v(\overline{r}_k; x_k) \geq \kappa_2 \kappa_4 \xi^2 \min\left\{\frac{1}{1+\Omega_{\bar{j}}}, \delta\right\} =: \zeta' > 0. \tag{4.50}$$

We may now combine (4.50), (4.48), and (4.47) to state that (4.7c) must be violated for sufficiently large $k \in \mathcal{K}''$ and, consequently, the penalty parameter will be decreased. However, this is a contradiction to (4.43), so we conclude that $F_{\text{FEAS}}(x_k) \to 0$. The fact that every limit point of $\{x_k\}_{k\geq 0}$ is an infeasible stationary point follows since $\|c_k\|_2 \geq t$ for all $k \geq \overline{k}$ from (4.41) and $F_{\text{FEAS}}(x_k) \to 0$. □

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

4.3.2 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 7 that

$$\lim_{j\to\infty} t_j = \lim_{j\to\infty} T_j = 0. \tag{4.51}$$

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. The proof of this lemma is the same as the proof of the Lemma 3.3.4 because the proof only depends on (4.51) and the mechanism of the update of the Lagrange multiplier vector which are the same for both lemmas. So we do not provide it here for the sake of brevity.

Lemma 4.3.5. *If $|\mathcal{Y}| = \infty$ and $\mu_k = \mu$ for some $\mu > 0$ for all sufficiently large k, then the following limits hold:*

$$\lim_{j\to\infty} c_{k_j} = 0 \tag{4.52a}$$

$$\text{and} \quad \lim_{j\to\infty} F_L(x_{k_j}, \widehat{y}_{k_j}) = 0. \tag{4.52b}$$

Thus, any limit point (x_*, y_*) of $\{(x_{k_j}, \widehat{y}_{k_j})\}_{j\geq 0}$ is first-order stationary for (2.1).

Finally, we consider the case when the penalty parameter converges to zero. In this case, we state the following lemma without showing a proof. The proof is the same as that of Lemma 3.3.6 because it only uses (4.51), the fact that the penalty parameter converges to zero and the mechanism of the update of the Lagrange multiplier vector, which are all the same for both lemmas.

Lemma 4.3.6. *If $|\mathcal{Y}| = \infty$ and $\mu_k \to 0$, then*

$$\lim_{k\to\infty} c_k = 0. \tag{4.53}$$

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

$$\lim_{j\in\mathcal{J}, j\to\infty} \|F_L(x_{k_j}, \widehat{y}_{k_j})\|_2 = 0 \quad \text{or} \quad \lim_{j\in\mathcal{J}, j\to\infty} \|F_L(x_{k_j}, \pi(x_{k_j}, y_{k_j-1}, \mu_{k_j-1}))\|_2 = 0. \tag{4.54}$$

In such cases, if the first (respectively, second) limit in (4.54) *holds, then along with* (4.53) *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* (2.1)*.*

4.3.3 Overall convergence result

Our main global convergence result for Algorithm 7 is as follows.

Theorem 4.3.1. *If Assumptions 4.2.1 and 4.3.1 hold, then one of the following holds:*

(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 (2.1)*; 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* (2.1).

Proof. Lemmas 4.3.3, 4.3.4, 4.3.5 and 4.3.6 cover the only four possible outcomes of Algorithm 7; the result follows from those described in these lemmas. □

Observe that the conclusions in Theorem 4.3.1 are the same as in Theorem 3.3.1. The comments following Theorem 3.3.1 discuss the consequences of these results. In particular, they suggest how Algorithm 7 may be modified to guarantee convergence to first-order stationary points, even in case (iii) of Theorem 4.3.1. However, as mentioned in those comments, we do not consider these modifications to the algorithm to have practical benefits. This perspective is supported by the numerical tests presented in Chapter 5.

Chapter 5

Numerical Tests on the Adaptive AL Methods

This chapter provides 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. In §5.1 we present the numerical results from our MATLAB implementation of both adaptive AL algorithms. Numerical results in §5.2 come from an implementation of a simple modification of the AL trust-region algorithm in the Fortran software package LANCELOT.[17]

5.1 Numerical experiments with a Matlab implementation

We implemented our adaptive AL line search algorithm, i.e., Algorithm 7, and the adaptive AL trust-region method given as Algorithm 5 in MATLAB. 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 thesis.

5.1.1 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 AL method given by Algorithm 1. We refer to these algorithms as `BAL-LS` (**b**asic **a**ugmented **L**agrangian **l**ine **s**earch) and `BAL-TR` (**b**asic **a**ugmented **L**agrangian **t**rust **r**egion), respectively. These algorithms differed in that `BAL-LS` used a line search and `BAL-TR` used a trust-region strategy for step acceptance. In addition, like Algorithm 7 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 5 and two variants of Algorithm 7. 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 (3.8) or (4.4), all algorithms 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 on the reduced space defined by the working set. During the PCG routine, if a 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, and 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), yielded an approximate solution that reduced the subproblem objective by at least as much as it was reduced by the Cauchy point, and, overall, 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.

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 6 was performed with (4.6) as stated (i.e., with $\tilde{q}$), but the above PCG routine to compute the search direction was applied to (4.4) *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 (3.4) or (4.1). Instead, we simply used the Cauchy steps as approximate solutions of these subproblems. Finally, we note that in the steering mechanism, we checked condition (4.7c) with the Cauchy steps for each subproblem, despite the fact that the search direction was computed as a more accurate solution of (3.8) or (4.4). 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.

For the computation of the estimates $\{\widehat{y}_{k+1}\}$ (which are required to satisfy (3.13)), 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

$$\|F_{\mathrm{L}}(x_k, y_k)\|_\infty \leq \kappa_{\mathrm{opt}} \quad \text{and} \quad \|c_k\|_\infty \leq \kappa_{\mathrm{feas}}, \tag{5.1}$$

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

$$\|F_{\mathrm{FEAS}}(x_k)\|_\infty \leq \kappa_{\mathrm{opt}}, \quad \|c_k\|_\infty > \kappa_{\mathrm{feas}}, \quad \text{and} \quad \mu_k \leq \mu_{\min}. \tag{5.2}$$

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 (5.1), which was always the preferred outcome. Each algorithm terminated with a message of failure if neither (5.1) nor (5.2) 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 5.1.

We close this subsection with a discussion of some additional differences between the algorithms as stated in this thesis 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

Table 5.1: 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}	γ_δ	0.5
δ_0	1	δ	10^4	δ_R	10^{-4}	Γ_δ	5/3

incorporate features that we found to work well in our experiments. First, while all algorithms use the input parameter γ_μ given in Table 5.1 for decreasing the penalty parameter, we decrease the penalty parameter less significantly in the steering mechanism. In particular, in line 20 of Algorithm 5 and line 20 of Algorithm 7, we replace γ_μ with 0.7. Second, in the line search algorithms, rather than set the trust-region radii as in (4.2) and (4.5) 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 (4.2) and (4.5) 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 (3.14). Fourth, we initialized the target values as

$$t_0 \leftarrow t_1 \leftarrow \max\{10^2, \min\{10^4, \|c_k\|_\infty\}\} \tag{5.3}$$

$$\text{and } T_1 \leftarrow \max\{10^0, \min\{10^2, \|F_{\text{L}}(x_k, y_k)\|_\infty\}\}. \tag{5.4}$$

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 19 of Algorithm 5 and line 19 of Algorithm 7 to be satisfied whenever $\mu_k \leq 10^{-4}$.

5.1.2 Results on the CUTEst test problems

We tested our MATLAB algorithms on the subset of problems from the CUTEST[52] collection that have at least one general constraint and at most 1000 variables and 1000 constraints. (We convert all general inequality constraints to equality constraints by using slack variables. Other approaches[24,53,54] use an AL function defined for the inequality constraints instead of introducing additional slack variables.) 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 (5.1) or (5.2) was satisfied, as opposed to reaching the maximum number of allowed iterations, which was set to 10^4. 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[55] 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 (α, β), then, on $100 \times \beta\%$ of

the problems, the number of iterations required by the algorithm was less than or equal to 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 5.1 and 5.2 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 5.1) or the number of function evaluations (Figure 5.2) 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 (4.8) (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.

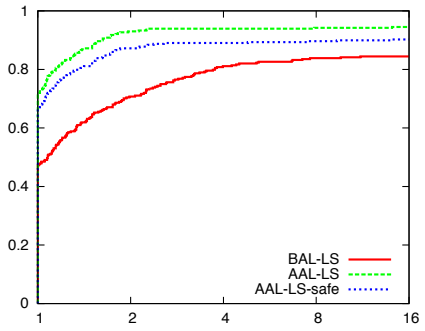

Figure 5.1: Performance profile for iterations: line search algorithms on the CUTEst set.

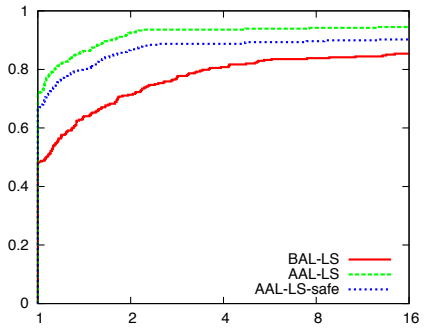

Figure 5.2: Performance profile for function evaluations: line search algorithms on the CUTEst set.

Figures 5.3 and 5.4 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 5.3) or the number of gradient evaluations (Figure 5.4) 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 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.

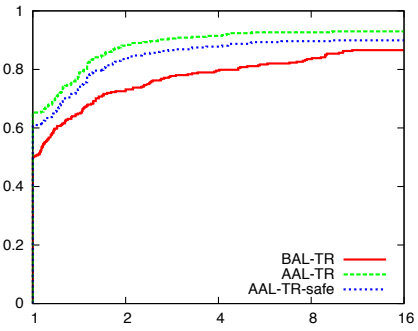

Figure 5.3: Performance profile for iterations: trust-region algorithms on the CUTEst set.

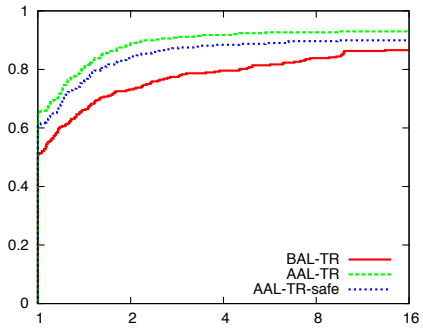

Figure 5.4: Performance profile for gradient evaluations: trust-region algorithms on the CUTEst set.

The performance profiles in Figures 5.1–5.4 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 5.2. 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 5.2. First, as may be expected, the algorithms that employ steering typically reduce the penalty parameter

Table 5.2: 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

below its initial value on some problems on which the other algorithms do not reduce it at all. This, in itself, is not a 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 5.2 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.

5.1.3 Results on the COPS test problems

We also tested our MATLAB software on the large-scale constrained problems available in the COPS[19] 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[1,2] provided in the collection. We removed problem *robot1* since algorithms `BAL-TR` and `AAL-TR` both encountered function evaluation errors. Additionally, the maximum time limit of 3600 seconds was reached by every solver on problems *chain*, *dirichlet*, *henon*, and *lane_emden*, so these problems were also 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.[56] 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*

$$r^j_{AB} := -\log_2(\mathrm{m}^j_A/\mathrm{m}^j_B), \quad \text{where} \quad \begin{cases} \mathrm{m}^j_A & \text{is the measure for } A \text{ on problem } j \\ \mathrm{m}^j_B & \text{is the measure for } B \text{ on problem } j. \end{cases} \tag{5.5}$$

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 5.5–5.8. For the same reasons as discussed in §5.1.2, 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 §5.1.2, 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 5.5 and 5.6 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 5.7 and 5.8 for the trust-region algorithms, the red bar for *catmix* indicates that `AAL-TR` failed on it.

The results in Figures 5.5 and 5.6 show that `AAL-LS` 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 5.7 and 5.8 that, despite the one failure, `AAL-TR` is generally superior to `BAL-TR`. We conclude, therefore, that steering was beneficial on this test set, especially in terms of the trust-region methods.

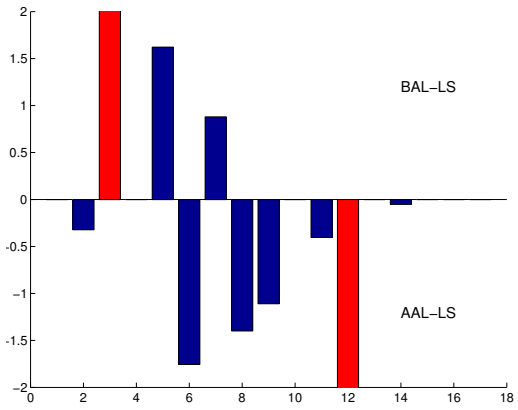

Figure 5.5: Outperforming factors for iterations: line search algorithms on the COPS set.

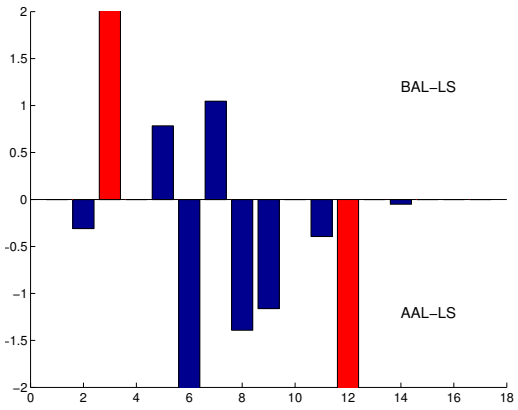

Figure 5.6: Outperforming factors for function evaluations: line search algorithms on the COPS set.

5.1.4 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.[20] 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 comprised of 28

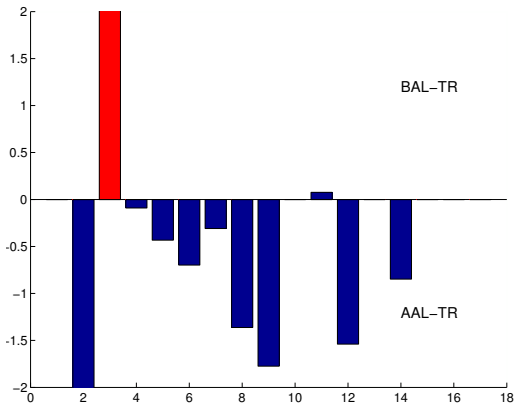

Figure 5.7: Outperforming factors for iterations: trust-region algorithms on the COPS set.

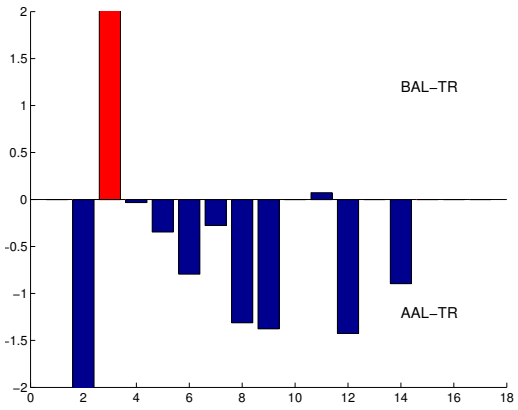

Figure 5.8: Outperforming factors for gradient evaluations: trust-region algorithms on the COPS set.

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 §5.1.3. Figures 5.9 and 5.10 reveal that `AAL-LS` typically outperforms `BAL-LS` in terms of both iterations and function evaluations, and Figures 5.11 and 5.12 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 §5.1.3. In any case, we have presented convincing numerical evidence that steering often has an overall beneficial effect on the performance of our MATLAB solvers.

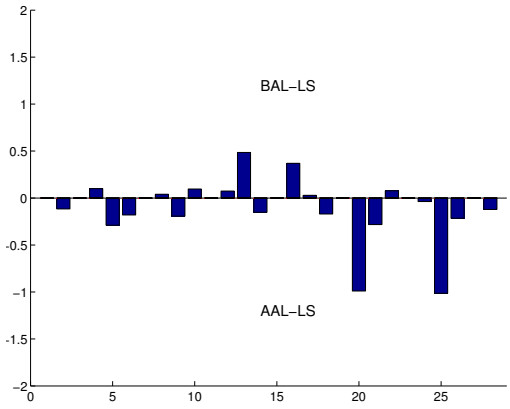

Figure 5.9: Outperforming factors for iterations: line search algorithms on OPF tests.

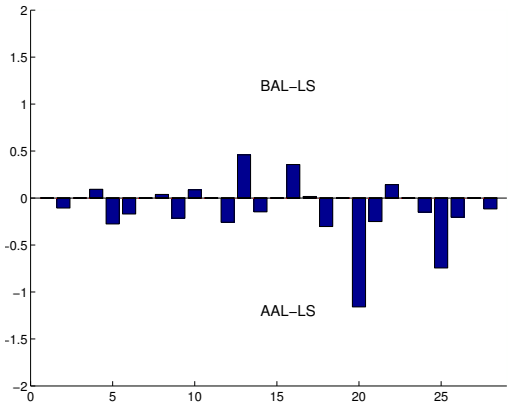

Figure 5.10: Outperforming factors for function evaluations: line search algorithms on OPF tests.

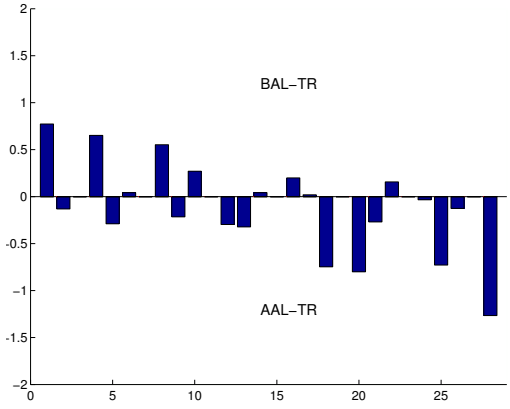

Figure 5.11: Outperforming factors for iterations: trust-region algorithms on OPF tests.

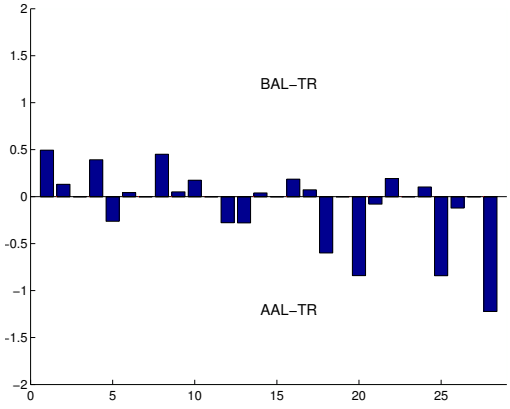

Figure 5.12: Outperforming factors for gradient evaluations: trust-region algorithms on OPF tests.

5.2 Numerical experiments with a Fortran implementation

We implemented a simple modification of the AL trust-region algorithm in the LANCELOT software package. 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 Algorithms 5 and our implemented algorithm in LANCELOT in terms of, e.g., the multiplier updates.

5.2.1 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 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 (4.7) 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 Algorithm 1 since they use optimality measures for both the Lagrangian

and the AL functions (see line 28 of Algorithm 7) rather than only that 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[17,57] package that is available in the GALAHAD library.[58] Specifically, we made three principle enhancements in LANCELOT. First, along the lines of the model q 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 (3.18a) or (4.9a). Second, we added routines to compute generalized Cauchy points[59] for both the constraint violation measure model q_v and $\hat{q}$ during the loop in which μ was decreased until the steering test (4.7c) was satisfied; recall the **while** loop starting on line 19 of Algorithm 7. Third, we

used the value for μ determined in the steering procedure to compute a generalized Cauchy point for the Newton model q_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[60] or via a backtracking Armijo search along the same arc.[49] 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 de-

creased by a factor of 0.7 until the steering test (3.11c) was satisfied. All other control parameters were set to their default `lancelot` values as given in its documentation. A problem was considered to be solved if `lancelot` returned the flag **status** = 0, which indicated that final constraint violation and norm of the projected gradient were less than 10^{-6}. We also considered a problem to be solved if `lancelot` returned the flag **status** = 3 (indicating that the trial step was too small to make any progress), the constraint violation was below 10^{-5}, and the norm of the projected gradient was less than 10^{-2}. Importantly, these criteria for deeming a problem to have been solved, were used by all three variants described above.

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.

5.2.2 Results on the 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 5.13 and 5.14, 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 influence that steering can have on performance.

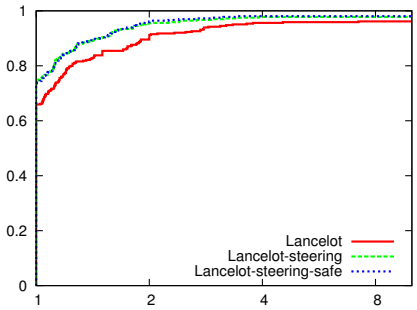

Figure 5.13: Performance profile for iterations: LANCELOT algorithms on the CUTEST set.

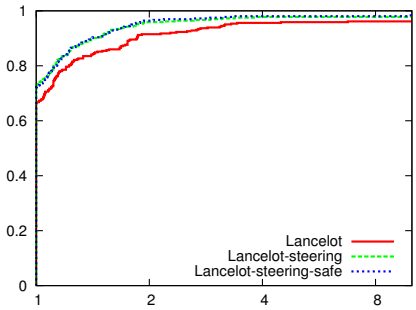

Figure 5.14: Performance profile for gradient evaluations: LANCELOT algorithms on the CUTEST set.

As in §5.1.2, 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 5.3.

We make a few remarks about the results in Table 5.3. 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 prob-

Table 5.3: 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

lems 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 5.3 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.

5.3 Conclusion

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 as well as a minimally modified variant of the well-known AL software LANCELOT. We feel confident that these numerical experiments clearly show the benefits of our new adaptive parameter updating strategy, in terms of efficiency and reliability.

Chapter 6

ADMM in Low Rank Subspace Clustering

6.1 Introduction

In this chapter, we study ADMM (see Algorithm 2) and its application in subspace clustering. Subspace clustering is a learning task that arises in many computer vision and pattern recognition applications such as motion segmentation [61], face clustering [62] and image processing [63].

In many problems—like the ones mentioned above—the high-dimensional data we observe lie approximately in a union of multiple low-dimensional subspaces where both the subspaces and the membership of data points to the subspaces are unknown. The task of subspace clustering is to identify the subspaces and uncover this

unknown membership, i.e., to partition the data by assigning each data point to its corresponding low-dimensional subspace.

Formally, let $X \in \mathbb{R}^{d\times n}$ denote a data matrix where each column is a data point in $\mathbb{R}^d$ obtained by adding noise and/or sparse errors to a clean data point drawn from one out of k unknown subspaces of dimensions $\{d_i\}_{i=1}^k$ with $d_i \ll d$. Subspace clustering seeks to cluster the potentially noisy data points into k groups such that the points generated from the same subspace are in the same group.

A number of methods have been developed to tackle the subspace clustering problem that include algebraic, iterative, statistical, and spectral clustering-based methods (see [64, 65] and the references therein). Among these methods, spectral clustering-based ones have shown superior performance (see [64]). They basically consist of two steps. The first step is to compute an affinity matrix from the data that encodes the similarity of each pair of data points. The second step is to construct a weighted graph using the affinity matrix and then apply spectral clustering ([66]) on the graph.

Many recent spectral clustering-based methods compute the affinity matrix based on the *self-expressiveness property* that was first proposed in [67]. A data matrix X is said to satisfy the self-expressiveness property if there exists $C \in \mathbb{R}^{n\times n}$ such that

$$X = XC \;\text{ and }\; \operatorname{diag}(C) = 0, \tag{6.1}$$

where $\operatorname{diag}(C)$ is a vector formed from the diagonal elements of C. Although many

such matrices C may exist, for the purpose of subspace clustering the ones satisfying $C_{ij} = 0$ if data points x_i and x_j are from different subspaces are of particular interest since they are good candidates for building the affinity matrix. To find such a C, existing methods solve the regularized optimization problem

$$\underset{C}{\text{minimize}} \ \|C\| \ \text{ subject to } \ X = XC \ \text{ and } \ \text{diag}(C) = 0. \tag{6.2}$$

For example, sparse subspace clustering uses the ℓ_1-norm to promote sparsity in the matrix C [see 33, 67–71]. In this chapter, we focus on the class of low rank subspace clustering (LRSC) methods, where the nuclear norm is used to encourage C to be low-rank.

Several papers have analyzed LRSC methods. The work of [72, 73] presented a convex formulation of LRSC for noiseless data and data with outliers. An alternative optimization framework that built upon nonconvex formulations of LRSC was presented in [74] for data contaminated by dense noise and/or sparse gross errors. Finally, a more general discussion on LRSC approaches, including the relationship between several LRSC methods, was presented in [75].

The contribution made by this chapter involves a nonconvex model formulations introduced in [74]. In particular, we prove that a carefully constructed alternating direction method of multipliers (ADMM) generates iterates that converge to a first-order solution to the nonconvex optimization problem suggested in [74], thus

answering an outstanding theoretical question. In the process, we reveal the critical threshold value for choosing the penalty parameter, whose existence was previously unknown. Numerical experiments on synthetic data and a face clustering data set are used to validate the algorithmic approach.

This chapter is organized as follows. In Section 6.2 we review LRSC in the case of noiseless data. We then turn to clustering data with noise in Section 6.3 when we describe the LRSC method by Vidal and Favaro [74]. In Section 6.4, we introduce an ADMM algorithm for solving the optimization problem used in the LRSC method described in Section 6.3. In Section 6.5, we establish convergence of the ADMM algorithm. Finally, in Section 6.6 we present numerical results comparing LRSC-ADMM to other low rank based spectral clustering methods on both synthetic data and a face clustering problem.

6.2 LRSC for clean data

Let $X \in \mathbb{R}^{d\times n}$ denote a data matrix where the j-th column x_j is a data point in $\mathbb{R}^d$. Suppose that the data points in X are drawn from a union of subspaces of unknown dimensions with each subspace containing a number of points that is larger than its dimension. LRSC for clean data performs subspace clustering on X using a

two-step procedure. The first step is to solve the following optimization problem:

$$\underset{C}{\text{minimize}} \ \|C\|_* \ \text{subject to} \ X = XC \ \text{ and } \ C = C^T, \tag{6.3}$$

where $\|C\|_* = \sum_i \sigma_i(C)$ is the nuclear norm with $\sigma_i(C)$ denoting the i-th singular value of C.

After computing the minimizer C^*, the second step of LRSC uses $|C^*|$ as an affinity matrix and applies spectral clustering; here, $|C^*|$ means taking the absolute value of each element of C^*.

We have the following two nice results about (6.3) (see [65] for more details).

Lemma 6.2.1. *Problem* (6.3) *has a unique solution given by*

$$C^* = V_1 V_1^T, \tag{6.4}$$

where the columns of V_1 are the right singular vectors from the compact singular value decomposition (SVD) of X, i.e., each column of V_1 is a right singular vector corresponding to a positive singular value of X. Moreover, the optimal value of problem (6.3) *is the rank of matrix X.*

The second result requires the following definition.

Definition 6.2.1 (independent subspaces)**.** *A collection of k subspaces $\{S_i\}_{i=1}^k$ is said to be* independent *if and only if the dimension of their sum is equal to the sum*

of their dimensions. Here, the sum for subspaces is defined as

$$S_1 \oplus S_2 \oplus \cdots \oplus S_k = \{x = x_1 + x_2 + \cdots + x_k : \ x_1 \in S_1, \ x_2 \in S_2, \ldots, x_k \in S_k\}.$$

We may now state the second result.

Theorem 6.2.1. *Let X be a matrix whose columns are drawn from a union of independent subspaces, where for each subspace, the number of points is larger than the dimension of the subspace. Also let $U_1\Sigma_1 V_1^T$ be the compact SVD of X and let $C = V_1 V_1^T$. Then each C_{ij}, the (i,j)-th entry of C, has the following property:*

$$C_{ij} = 0 \quad \textit{if points } x_i \textit{ and } x_j \textit{ are from different subspaces.} \tag{6.5}$$

When the matrix C satisfies (6.5)*, it is said to be a subspace preserving representation.*

In summary, when the data points are clean, the subspaces are independent, and sufficiently many data points are drawn from each subspace, then the solution to (6.3) can be used to build an affinity matrix that is appropriate for spectral clustering.

6.3 LRSC for noisy data

We have seen that the optimization problem (6.3) can be very useful in subspace clustering of clean data. However, in practice, data may be contaminated by dense noise, sparse gross error, or both. Therefore, people have come up with generaliza-

tions of problem (6.3) that account for noise. For the case when there is only small dense noise or only sparse errors, a couple of optimization formulations with closed-form solutions or tractable algorithms have been proposed that show good empirical performance; these methods will not be discussed further in this thesis, but we refer the reader to [65] for more details.

In this section, we focus on the case where the data is corrupted by both small dense noise and sparse gross error. To be more precise, we assume that the observed data matrix can be written as $X = B + G + E$, the sum of an unknown clean data matrix $B \in \mathbb{R}^{d\times n}$, an unknown noise matrix $G \in \mathbb{R}^{d\times n}$ that is dense with elements that are small in magnitude, and an unknown error matrix $E \in \mathbb{R}^{d\times n}$ that is sparse with a number of nonzero elements that is a small relative to $d \times n$ but whose nonzero values may be large in magnitude. Also, in the spirit of LRSC, we assume that the columns of the clean matrix B are drawn from a union of low dimensional subspaces.

One LRSC method, introduced in [74], is based on solving the nonconvex optimization problem

$$\begin{aligned} &\underset{B,C,E,G}{\text{minimize}} \;\; \|C\|_* + \frac{\alpha}{2}\|G\|_F^2 + \gamma\|E\|_1 \\ &\text{subject to} \;\; X = B + G + E, \;\; B = BC, \;\; \text{and} \;\; C = C^T, \end{aligned} \tag{6.6}$$

where $\|G\|_F = \sqrt{\sum_{ij} G_{ij}^2}$, and $\|E\|_1 = \sum_{ij} |E_{ij}|$ are, respectively, the Frobenius and ℓ_1 norms and $\alpha > 0$, $\gamma > 0$ are weighting parameters. After solving problem (6.6) and obtaining an optimal C^*, the affinity matrix is defined as $|C^*| + |C^{*T}|$, and then

spectral clustering is applied. In section 6.6, we will discuss a way to solve (6.6) and present numerical results for this method.

Our main theoretical results in this chapter are developed around another LRSC method proposed by [74], which computes an affinity matrix by solving the following nonconvex optimization problem

$$\begin{aligned} &\underset{A,C,E}{\text{minimize}} \;\; \|C\|_* + \frac{\tau}{2}\|A - AC\|_F^2 + \gamma\|E\|_1 \\ &\text{subject to} \;\; X = A + E \;\; \text{and} \;\; C = C^T, \end{aligned} \tag{6.7}$$

where $\tau > 0$ and $\gamma > 0$ are weighting parameters. As before, after solving (6.7) and obtaining an optimal C^*, we define the affinity matrix as $|C^*| + |C^{*T}|$, and then apply spectral clustering. One can motivate (6.7) from (6.6) as follows. We can fisrt consider using $A = B + G$ in (6.6). Then minimizing the term $\|A - AC\|_F^2$, in lieu of enforcing $B = BC$ explicitly, makes sense because the condition $B = BC$ implies that $(B+G)C = B+G+GC-G$, which after using $A = B+G$ yields $AC = A+GC-G$. Thus, when elements of G are small, minimizing $\|A - AC\|_F^2$ is appropriate.

In the next two sections, we develop an ADMM algorithm for solving (6.7) and provide a detailed convergence analysis for the algorithm.

6.4 ADMM for LRSC (LRSC-ADMM)

We first make use of a result from [74, Theorem 1] to show that problem (6.7) is equivalent to a problem that only involves two variables A and E. Specifically, the result is that for a fixed A, the problem

$$\underset{C}{\text{minimize}} \;\; \|C\|_* + \frac{\tau}{2}\|A - AC\|_F^2 \;\; \text{subject to} \;\; C = C^T \tag{6.8}$$

has a closed-form minimizer $C^* = V\mathcal{P}_\tau(\Sigma)V^T$, where $A = U\Sigma V^T$ and the operator $\mathcal{P}$ acts on each element of Σ as

$$\mathcal{P}_\tau(x) = \begin{cases} 1 - \frac{1}{\tau x^2} & \text{if } x > 1/\sqrt{\tau}, \\ 0 & \text{if } x \leq 1/\sqrt{\tau}. \end{cases} \tag{6.9}$$

Moreover, the optimal value of the objective function for problem (6.8) is

$$\Phi_\tau(A) := \sum_{i=1}^{\min(d,n)} \phi(\sigma_i) \tag{6.10}$$

with σ_i denoting the i-th singular value of A,

$$\begin{aligned} \phi(\sigma_i) &:= (1 - \frac{1}{2\tau}\sigma_i^{-2})\mathbb{1}_{\mathcal{I}_1}(\sigma_i) + \frac{\tau}{2}\sigma_i^2\mathbb{1}_{\mathcal{I}_2}(\sigma_i), \\ \mathcal{I}_1 &:= \left(\frac{1}{\sqrt{\tau}}, +\infty\right), \text{ and} \\ \mathcal{I}_2 &:= \left[0, \frac{1}{\sqrt{\tau}}\right], \end{aligned} \tag{6.11}$$

where $\mathbb{1}_{\mathcal{I}}$ is the indicator function

$$\mathbb{1}_{\mathcal{I}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases}$$

Using this result, we can see that solving problem (6.7) is equivalent to first solving

$$\begin{aligned} \underset{A,E}{\text{minimize}} \;\; & \Phi_\tau(A) + \gamma\|E\|_1 \\ \text{subject to} \;\; & X = A + E \end{aligned} \tag{6.12}$$

to get (A^*, E^*) and then setting $C^* \leftarrow V\mathcal{P}_\tau(\Sigma)V^T$, where $A^* = U\Sigma V^T$ and $\mathcal{P}_\tau$ is the operator defined in (6.9). This is the key problem that we adopt when using ADMM to perform LRSC on noisy data, as we describe next.

Given a noisy data matrix X, we use ADMM to solve problem (6.12). Such a method uses the augmented Lagrangian function, which for problem (6.12), is

$$\mathcal{L}(A, E, Y) = \Phi_\tau(A) + \gamma\|E\|_1 + \langle Y, X - A - E\rangle + \frac{\rho}{2}\|X - A - E\|_F^2. \tag{6.13}$$

where the inner product is defined as $\langle Y, X - A - E\rangle = \text{trace}\left(Y^T(X - A - E)\right)$.

The ADMM scheme for LRSC, which we call LRSC-ADMM and is stated as Algorithm 8, essentially performs one pass of an alternating block-wise minimization scheme per iteration, followed by an update to the multipliers (see Section 2.2 for a review of ADMM). The solution that results from LRSC-ADMM is the triple

(E^*, A^*, Y^*). From this triple, we attain an affinity matrix $|C^*| + |C^{*T}|$, where $C^* = V\mathcal{P}_\tau(\Sigma)V^T$, $A^* = U\Sigma V^T$, and $\mathcal{P}_\tau$ is defined in (6.9). Finally, spectral clustering is performed using this affinity matrix.

Algorithm 8 ADMM for LRSC (LRSC-ADMM)

1: Choose initial values for A_0, Y_0, and $\rho > 0$.
2: **while** the stopping criterion is not satisfied **do**
3: $\quad E_{k+1} = \operatorname{argmin}_E \gamma\|E\|_1 + \langle Y_k, X - A_k - E\rangle + \frac{\rho}{2}\|X - A_k - E\|_F^2$,
4: $\quad A_{k+1} = \operatorname{argmin}_A \Phi_\tau(A) + \langle Y_k, X - A - E_{k+1}\rangle + \frac{\rho}{2}\|X - A - E_{k+1}\|_F^2$,
5: $\quad Y_{k+1} = Y_k + \rho(X - A_{k+1} - E_{k+1})$.
6: **end while**

We make two remarks about Algorithm 8. First, the optimization subproblems on lines 3 and line 4 have closed-form solutions, which makes Algorithm 8 easy to implement (see [74] for the exact formulas). Second, the order of the subproblems used on lines 3 and 4 of Algorithm 8 matters in our case, a fact that we make clear in the next section when we analyze the convergence of LRSC-ADMM.

6.5 Convergence of LRSC-ADMM

Except for the fact that the variables in (6.12) are matrices, problem (6.12) has the same form as (2.14) after a simple change of variable. (Simply regard $X - E$ as the new variable.) Therefore, if we can show that the assumptions in Theorem 2.2.1 are satisfied, then we automatically obtain a convergence result since the proof of that theorem can be carried over to the matrix case. However, for completeness, we still present the entire convergence analysis of Algorithm 8 without using the change of

variable mentioned above. We should also point out that we provide two new results, namely Lemma 6.5.3 and Lemma 6.5.6. Lemma 6.5.3 gives a sufficient condition for the second assumption in Theorem 2.2.1 to hold. Lemma 6.5.6 is specific to our problem formulation that ensures that the iterates computed from Algorithm 8 will have at least one limit point, which is not otherwise guaranteed by Theorem 2.2.1.

Our convergence result relies on the following two properties of the function $\Phi_\tau(\cdot)$.

Proposition 6.5.1. *The function Φ_τ is differentiable and its gradient, at any $A \in \mathbb{R}^{d\times n}$, is given by*

$$\nabla\Phi_\tau(A) = U\mathrm{Diag}([\phi'(\sigma_1),\ldots,\phi'(\sigma_r)])V^T, \tag{6.14}$$

where $r = \min(d,n)$, $U \in \mathbb{R}^{d\times r}$ and $V \in \mathbb{R}^{n\times r}$ are from the SVD $A = U\Sigma V^T$ with $\Sigma = \mathrm{Diag}([\sigma_1,\ldots,\sigma_r])$ and σ_i the i-th singular value of A.

Proof. According to the definition of Φ_τ in (6.10), for any $A \in \mathbb{R}^{d\times n}$, $\Phi_\tau(A)$ can be regarded as a function of the singular values of A, i.e., $\Phi_\tau(A) = f(\sigma(A))$, where $\sigma(A)$ denotes the vector of singular values of A and $f : \mathbb{R}^r \to \mathbb{R}$ is defined as $f(v) = \sum_i \tilde{\phi}(v_i)$ for any $v \in \mathbb{R}^r$, where $\tilde{\phi}(v_i) := \phi(v_i)$ if $v_i \geq 0$ and $\tilde{\phi}(v_i) := \phi(-v_i)$ if $v_i < 0$.

It is not difficult to see that all the partial derivatives of f exist and are continuous on $\mathbb{R}^r$. Therefore f is continuously differentiable on $\mathbb{R}^r$ (see [76, Theorem 9.21]). Then by [77, Corollary 7.4], Φ_τ is also differentiable. The gradient of Φ_τ can be computed by applying [77, Theorem 7.1], which is the desired result. □

We notice that since $\phi'(0) = 0$, (6.14) can be simplified to

$$\nabla\Phi_\tau(A) = U_1 \mathrm{Diag}([\phi'(\sigma_1), \ldots, \phi'(\sigma_k)])V_1^T,$$

where $k = \mathrm{rank}(A)$, $U_1 \in \mathbb{R}^{d\times k}$ and $V_1 \in \mathbb{R}^{n\times k}$ are from the compact SVD $A = U_1\Sigma_1 V_1^T$ with $\Sigma_1 = \mathrm{Diag}([\sigma_1, \ldots, \sigma_k])$ and σ_i the i-th nonzero singular value of A.

Proposition 6.5.2. *Let $\nabla\Phi_\tau(A)$ be computed as in* (6.14)*. If ϕ' is Lipschitz continuous, then $\nabla\Phi_\tau$ is Lipschitz continuous with the same Lipschitz constant.*

The proof of this proposition is a simple modification of that of [78, Theorem 1.1]. However, we show the details below for completeness. To show this proposition, we need the following two results from [78], which we state without proofs.

Lemma 6.5.1 ([78, Proposition 4.1])**.** *Let $f : \mathbb{R}_+ \to \mathbb{R}$ be Lipschitz continuous with $f(0) = 0$ where $\mathbb{R}_+$ denotes the set of nonnegative real numbers. Let $\|f\|_{Lip}$ denote the smallest Lipschitz constant of f:*

$$\|f\|_{Lip} := \sup_{x_1,x_2\in\mathbb{R}_+} \frac{|f(x_1) - f(x_2)|}{|x_1 - x_2|}$$

and define the following:

$$\|f\|_{Lip-\mathbb{C}} := \sup_{x_1,x_2\in\mathbb{R}_+,c\in\mathbb{T}} \frac{|f(x_1) - cf(x_2)|}{|x_1 - cx_2|},$$

where $\mathbb{T}$ denotes the unit circle in $\mathbb{C}$, the space of complex numbers. Then the following

equality holds:

$$\|f\|_{Lip-\mathbb{C}} = \|f\|_{Lip}.$$

Before we state the second result, we introduce a few definitions. We say that a complex square matrix is complex doubly substochastic if the ℓ_1 norm of each row and column is less than or equal to 1. Let π denote any permutation of length $r = \min(d, n)$ and let ν be a vector in $\mathbb{C}^r$ containing unimodular entries. We denote by $M_{\pi,\nu}$ the $r \times r$ matrix whose (j, π_j)-th value is ν_j , all other entries zero. Then we have the following lemma.

Lemma 6.5.2 ([78, Lemma 3.1])**.** *An $r \times r$ matrix is complex doubly substochastic if and only if it lies in the convex hull of $\{M_{\pi,\nu} : \pi, \nu\}$.*

We now proceed to prove Proposition 6.5.2.

Proof of Proposition 6.5.2. We will show that for $d \times n$ real matrices A and B:

$$\|\nabla\Phi_\tau(A) - \nabla\Phi_\tau(B)\|_F \leq \|\phi'\|_{Lip-C}\|A - B\|_F. \tag{6.15}$$

Proposition 6.5.2 is an immediate corollary of (6.15) and Lemma 6.5.1.

Let $r = \min(d, n)$. We write down the singular values decompositions of A and B

$$A = U_A\Sigma_A V_A^T \quad \text{and} \quad B = U_B\Sigma_B V_B^T,$$

where U_A and U_B are $d \times r$, Σ_A and Σ_B are $r \times r$, and V_A and V_B are $n \times r$. It follows

that the quantity $\|A - B\|_F^2$ can be written as

$$\begin{aligned}\|A - B\|_F^2 &= \|A\|_F^2 + \|B\|_F^2 - 2\langle A, B\rangle \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\left\langle U_A\Sigma_A V_A^T, U_B\Sigma_B V_B^T\right\rangle \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\left\langle U_B^T U_A\Sigma_A, \Sigma_B V_B^T V_A\right\rangle \\ &= \|A\|_F^2 + \|B\|_F^2 - 2\sum_{ae} \Sigma_B V_B^T V_A \odot U_B^T U_A \Sigma_A,\end{aligned}$$

where $\sum_{ae} M$ denotes the operation of summing all entries of the matrix M and $\odot$ denotes the Hadamard product.

$V_B^T V_A \odot U_B^T U_A$ is complex doubly substochastic because the two-norm of each row and column of $V_B^T V_A$ and $U_B^T U_A$ is less than or equal to 1. Therefore Lemma 6.5.2 implies that there exist $c_1, \ldots, c_m$ in $[0, 1]$ satisfying $\sum_{i=1}^m c_i = 1$, permutations π_i and length-r complex vectors ν_i with unimodular entries, $1 \leq i \leq m$, such that

$$V_B^T V_A \odot U_B^T U_A = \sum_{i=1}^m c_i M_{\pi_i,\nu_i}.$$

It then follows that

$$\begin{aligned}
\|A-B\|_F^2 &= \|A\|_F^2 + \|B\|_F^2 - 2\sum_{i=1}^{m} c_i \sum_{ae} \Sigma_B M_{\pi_i,\nu_i} \Sigma_A \\
&= \|A\|_F^2 + \|B\|_F^2 - 2\mathrm{Re}\left(\sum_{i=1}^{m} c_i \sum_{ae} \Sigma_B M_{\pi_i,\nu_i} \Sigma_A\right) \\
&= \sum_{i=1}^{m} c_i \left(\sum_{j=1}^{r} \sigma_j(A)^2 + \sum_{j=1}^{r} \sigma_j(B)^2 - 2\mathrm{Re}\left(\sum_{j=1}^{r} \sigma_j(B)[\nu_i]_j \sigma_{[\pi_i]_j}(A)\right)\right) \\
&= \sum_{i=1}^{m} c_i \sum_{j=1}^{r} \left|\sigma_j(B) - [\nu_i]_j \sigma_{[\pi_i]_j}(A)\right|^2,
\end{aligned}$$

where $[\nu_i]_j$ is the j-th element of ν_i, $[\pi_i]_j$ is the j-th element of π_i and $\mathrm{Re}(c)$ denotes the real part of a complex number c. The second equality above holds simply because $\sum_{i=1}^{m} c_i \sum_{ae} \Sigma_B M_{\pi_i,\nu_i} \Sigma_A$ does not have an imaginary part.

Using similar arguments, it is not hard to see that the above equation also applies to $\nabla\Phi_\tau(A)$ and $\nabla\Phi_\tau(B)$. Thus

$$\|\nabla\Phi_\tau(A) - \nabla\Phi_\tau(B)\|_F^2 = \sum_{i=1}^{m} c_i \sum_{j=1}^{r} \left|\phi'(\sigma_j(B)) - [\nu_i]_j \phi'(\sigma_{[\pi_i]_j}(A))\right|^2.$$

Using the definition of $\|\phi'\|_{Lip-\mathbb{C}}$ in Lemma 6.5.1, we have that

$$\begin{aligned}
\|\nabla\Phi_\tau(A) - \nabla\Phi_\tau(B)\|_F^2 &\le \|\phi'\|_{Lip-\mathbb{C}}^2 \sum_{i=1}^{m} c_i \sum_{j=1}^{r} \left|\sigma_j(B) - [\nu_i]_j \sigma_{[\pi_i]_j}(A)\right|^2 \\
&= \|\phi'\|_{Lip-\mathbb{C}}^2 \|A-B\|_F^2,
\end{aligned}$$

which is precisely (6.15). □

It is not difficult to show that $\phi'(\sigma_i) = \frac{1}{\tau}\sigma_i^{-3}\mathbb{1}_{\mathcal{I}_1}(\sigma_i) + \tau\sigma_i\mathbb{1}_{\mathcal{I}_2}(\sigma_i)$ is Lipschitz continuous with Lipschitz constant 3τ. Thus, Proposition 6.5.2 gives the following.

Corollary 6.5.1. *The function $\nabla\Phi_\tau(\cdot)$ is Lipschitz continuous with constant $L := 3\tau$.*

Having shown that Φ_τ is differentiable with Lipschitz continuous gradient, we now follow the ideas in [45] to show that any limit point of the sequence generated by Algorithm 8 is a first-order solution. To this end, we make the following assumption throughout the remainder of this section.

Assumption 6.5.1. *The penalty parameter of Augmented Lagrangian* (6.13) *is chosen to satisfy $\rho > 2L \equiv 6\tau$.*

Our next lemma establishes strong convexity of the subproblem in the A-update of Algorithm 8.

Lemma 6.5.3. *For all k, the function $\mathcal{L}(A, E_{k+1}, Y_k)$ being minimized in line 4 of Algorithm 8 is strongly convex with modulus $\xi(\rho)$ that satisfies $\xi(\rho) \geq \rho - L > L > 2L^2/\rho$.*

Proof. To show $\mathcal{L}(A, E_{k+1}, Y_k)$ is strongly convex in A, it suffices to show that $\Phi_\tau(A) + \frac{\rho}{2}\|A\|_F^2$ is strongly convex in A. From the Lipschitz continuity of $\nabla\Phi_\tau(\cdot)$ and [50, Theorem 3.1.4], we have that for any $d\times n$ matrices A_1 and A_2, the following inequality holds:

$$|\Phi_\tau(A_1) - \Phi_\tau(A_2) - \langle\nabla\Phi_\tau(A_2), A_1 - A_2\rangle| \leq \frac{L}{2}\|A_1 - A_2\|_F^2.$$

Rearranging terms gives us

$$\Phi_\tau(A_1) \geq \Phi_\tau(A_2) + \langle \nabla\Phi_\tau(A_2), A_1 - A_2\rangle - \frac{L}{2}\|A_1 - A_2\|_F^2. \tag{6.16}$$

Adding $\frac{\rho}{2}\|A_1\|_F^2$ to both sides and using basic algebra, we get

$$\begin{aligned}
&\Phi_\tau(A_1) + \frac{\rho}{2}\|A_1\|_F^2 && (6.17)\\
&\geq \Phi_\tau(A_2) + \langle \nabla\Phi_\tau(A_2), A_1 - A_2\rangle - \frac{L}{2}\|A_1 - A_2\|_F^2 + \frac{\rho}{2}\|A_1\|_F^2 && (6.18)\\
&= \Phi_\tau(A_2) + \frac{\rho}{2}\|A_2\|_F^2 + \langle \nabla\Phi_\tau(A_2) + \rho A_2, A_1 - A_2\rangle && (6.19)\\
&\qquad - \frac{\rho}{2}\|A_2\|_F^2 - \langle \rho A_2, A_1 - A_2\rangle - \frac{L}{2}\|A_1 - A_2\|_F^2 + \frac{\rho}{2}\|A_1\|_F^2 \\
&= \Phi_\tau(A_2) + \frac{\rho}{2}\|A_2\|_F^2 + \langle \nabla\Phi_\tau(A_2) + \rho A_2, A_1 - A_2\rangle + \frac{\rho - L}{2}\|A_1 - A_2\|_F^2. && (6.20)
\end{aligned}$$

The claims of the lemma follow from the above inequality and Assumption 6.5.1. □

The following lemma shows that the value of the augmented Lagrangian function (6.13) is non-increasing over the sequence of iterates.

Lemma 6.5.4. *For each k, we have that*

$$\mathcal{L}(A_{k+1}, E_{k+1}, Y_{k+1}) - \mathcal{L}(A_k, E_k, Y_k) \leq -\kappa(\rho)\|A_{k+1} - A_k\|_F^2 - \frac{\rho}{2}\|E_{k+1} - E_k\|_F^2,$$

where $\kappa(\rho) := \left(\frac{\xi(\rho)}{2} - \frac{L^2}{\rho}\right) > 0$.

Proof. The fact that $\kappa(\rho) > 0$ follows from Lemma 6.5.3. The optimality conditions

for the problem in line 4 of Algorithm 8 is given by

$$\nabla\Phi_\tau(A_{k+1}) - Y_k - \rho(X - A_{k+1} - E_{k+1}) = 0. \tag{6.21}$$

If we combine this result with line 5 of Algorithm 8, we get that

$$\nabla\Phi_\tau(A_{k+1}) = Y_{k+1}. \tag{6.22}$$

Using the Lipschitz continuity of $\nabla\Phi_\tau(\cdot)$, we have that

$$\|Y_{k+1} - Y_k\|_F = \|\nabla\Phi_\tau(A_{k+1}) - \nabla\Phi_\tau(A_k)\|_F \leq L\|A_{k+1} - A_k\|_F. \tag{6.23}$$

We now split the difference of the augmented Lagrangian as

$$\begin{aligned}
&\mathcal{L}(A_{k+1}, E_{k+1}, Y_{k+1}) - \mathcal{L}(A_k, E_k, Y_k) \\
&= \mathcal{L}(A_{k+1}, E_{k+1}, Y_{k+1}) - \mathcal{L}(A_{k+1}, E_{k+1}, Y_k) \\
&\quad + \mathcal{L}(A_{k+1}, E_{k+1}, Y_k) - \mathcal{L}(A_k, E_{k+1}, Y_k) \\
&\quad + \mathcal{L}(A_k, E_{k+1}, Y_k) - \mathcal{L}(A_k, E_k, Y_k).
\end{aligned} \tag{6.24}$$

The first term on the right-hand side of the equation can be bounded as

$$\begin{aligned} & \mathcal{L}(A_{k+1}, E_{k+1}, Y_{k+1}) - \mathcal{L}(A_{k+1}, E_{k+1}, Y_k) \\ = \; & \langle Y_{k+1} - Y_k, X - A_{k+1} - E_{k+1} \rangle \\ = \; & \frac{1}{\rho} \|Y_{k+1} - Y_k\|_F^2 \\ \leq \; & \frac{L^2}{\rho} \|A_{k+1} - A_k\|_F^2, \end{aligned} \tag{6.25}$$

where the first equality follows from definition (6.13), the second equality follows from line 5 of Algorithm 8, and the inequality follows from (6.23).

The second term on the right-hand side of equation (6.24) can be bounded as

$$\begin{aligned} & \mathcal{L}(A_{k+1}, E_{k+1}, Y_k) - \mathcal{L}(A_k, E_{k+1}, Y_k) \\ \leq \; & \langle \nabla_A \mathcal{L}(A_{k+1}, E_{k+1}, Y_k), A_{k+1} - A_k \rangle - \frac{\xi(\rho)}{2} \|A_{k+1} - A_k\|_F^2 \\ = \; & -\frac{\xi(\rho)}{2} \|A_{k+1} - A_k\|_F^2, \end{aligned} \tag{6.26}$$

where the inequality follows from Lemma 6.5.3 and the equality follows from the fact that $\nabla_A \mathcal{L}(A_{k+1}, E_{k+1}, Y_k) = 0$.

It is clear that $\mathcal{L}(A_k, E, Y_k)$ is a strongly convex function in E with modulus ρ. Also, since E_{k+1} minimizes $\mathcal{L}(A_k, E, Y_k)$, we have that for all k, that

$$0 \in \partial_E \mathcal{L}(A_k, E_{k+1}, Y_k).$$

Therefore, we can use similar arguments to bound the third term on the right-hand side of equation (6.24) as

$$\mathcal{L}(A_k, E_{k+1}, Y_k) - \mathcal{L}(A_k, E_k, Y_k) \leq -\frac{\rho}{2}\|E_{k+1} - E_k\|_F^2. \tag{6.27}$$

Combining the results in (6.25), (6.26) and (6.27) finishes the proof of the lemma. □

The next lemma shows that the augmented Lagrangian is not only non-increasing at each iteration, but that it also converges.

Lemma 6.5.5. *The sequence* $\{\mathcal{L}(A_k, E_k, Y_k)\}$ *generated by Algorithm 8 converges.*

Proof. By Lemma 6.5.4, the sequence $\{\mathcal{L}(A_k, E_k, Y_k)\}$ is monotonically non-increasing. So it suffices to prove that $\{\mathcal{L}(A_k, E_k, Y_k)\}$ is bounded below. This fact follows since

$$\begin{aligned}
&\mathcal{L}(A_k, E_k, Y_k) \\
&= \gamma\|E_k\|_1 + \Phi_\tau(A_k) + \langle Y_k, X - E_k - A_k\rangle + \frac{\rho}{2}\|X - E_k - A_k\|_F^2 \\
&= \gamma\|E_k\|_1 + \Phi_\tau(A_k) + \langle \nabla\Phi_\tau(A_k), X - E_k - A_k\rangle + \frac{\rho}{2}\|X - E_k - A_k\|_F^2 \\
&\geq \gamma\|E_k\|_1 + \Phi_\tau(X - E_k) \geq 0,
\end{aligned}$$

where the second equality follows from (6.22), the first inequality uses the Lipschitz continuity of $\nabla\Phi_\tau(\cdot)$ and Assumption 6.5.1, and the last inequality follows from the fact that both $\|\cdot\|_1$ and $\Phi_\tau(\cdot)$ are bounded below by zero. □

The next lemma is not in the original analysis of [45] and is specific to our problem.

It shows that the sequence of iterates $\{(A_k, E_k, Y_k)\}$ generated by Algorithm 8 is bounded, thus guaranteeing the existence of at least one limit point.

Lemma 6.5.6. *The sequences $\{A_k\}$, $\{E_k\}$, and $\{Y_k\}$ from Algorithm 8 are bounded.*

Proof. Notice that from Lemma 6.5.4, Lemma 6.5.5, and (6.23), we have

$$\begin{aligned} \lim_{k\to\infty} \|A_{k+1} - A_k\|_F &= 0, \\ \lim_{k\to\infty} \|E_{k+1} - E_k\|_F &= 0, \quad \text{and} \\ \lim_{k\to\infty} \|Y_{k+1} - Y_k\|_F &= 0. \end{aligned} \tag{6.28}$$

To show that $\{Y_k\}$ is bounded we note that line 3 of Algorithm 8 has a closed-form solution given by

$$E_{k+1} = \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1}Y_k),$$

where $\mathcal{S}_\epsilon(X)$ is the shrinkage thresholding operator defined by $[\mathcal{S}_\epsilon(X)]_{i,j} = sgn(X_{i,j}) \cdot \max(|X_{i,j}| - \epsilon, 0)$ for any $\epsilon > 0$ and matrix X. Plugging the closed-form expression for E_{k+1} into line 5 of the algorithm, yields

$$\begin{aligned} Y_{k+1} &= Y_k + \rho\left(X - A_{k+1} - \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1}Y_k)\right) \\ &= \rho\left(X - A_{k+1} + \rho^{-1}Y_k - \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1}Y_k)\right) \\ &= \rho\left(A_k - A_{k+1} + X - A_k + \rho^{-1}Y_k - \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1}Y_k)\right). \end{aligned}$$

Then, by applying the triangle inequality we find that

$$\|Y_{k+1}\|_F \leq \rho \left(\|A_k - A_{k+1}\|_F + \|X - A_k + \rho^{-1} Y_k - \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1} Y_k)\|_F \right).$$

We know from (6.28) that $\|A_k - A_{k+1}\|_F$ converges to zero. We also have from the definition of the operator $\mathcal{S}_{\gamma\rho^{-1}}$ that $\|X - A_k + \rho^{-1} Y_k - \mathcal{S}_{\gamma\rho^{-1}}(X - A_k + \rho^{-1} Y_k)\|_F$ is smaller than a constant because $\mathcal{S}_{\gamma\rho^{-1}}$ can change each element by at most $\gamma\rho^{-1}$. Therefore we get the boundedness of the sequence $\{Y_k\}$.

We now show the boundedness of $\{E_k\}$. From (6.13) we have that

$$\begin{aligned} \mathcal{L}(A_k, E_k, Y_k) &= \Phi_\tau(A_k) + \gamma\|E_k\|_1 + \frac{\rho}{2}\|X - A_k - E_k + \rho^{-1} Y_k\|_F^2 - \frac{1}{2\rho}\|Y_k\|_F^2 \\ &\geq \gamma\|E_k\|_1 - \frac{1}{2\rho}\|Y_k\|_F^2. \end{aligned} \tag{6.29}$$

Recall that we already proved that $\{Y_k\}$ is bounded, and from Lemma 6.5.5 we know that $\{\mathcal{L}(A_k, E_k, Y_k)\}$ is bounded. Therefore, it follows from (6.29) that the sequence $\{E_k\}$ has to be bounded.

Finally, we show that $\{A_k\}$ is bounded. By (6.28), we have $\|Y_{k+1} - Y_k\|_F \to 0$. It follows from line 5 of Algorithm 8 that $\|X - A_{k+1} - E_{k+1}\|_F \to 0$. Thus, the boundedness of $\{A_k\}$ follows from the boundedness of $\{E_k\}$. □

We are ready to use the previous results to show the final convergence theorem.

Theorem 6.5.1. *Let* (A^*, E^*, Y^*) *denote any limit point of the sequence* $\{(A_k, E_k, Y_k)\}$

generated by Algorithm 8. Then (A^*, E^*, Y^*) *is a first-order stationary point for problem* (6.12)*, i.e., it satisfies*

$$0 = \nabla\Phi_\tau(A^*) - Y^*, \tag{6.30}$$

$$0 \in \gamma\partial\|E^*\|_1 - Y^*, \quad \textit{and} \tag{6.31}$$

$$X = A^* + E^*. \tag{6.32}$$

Proof. From Lemma 6.5.6, we have that $\{(A_k, E_k, Y_k)\}$ has at least one convergent subsequence. Let $\{(A_{k_j}, E_{k_j}, Y_{k_j})\}_{j\geq 0}$ be a subsequence that converges to (A^*, E^*, Y^*). Then (6.30) is clearly true from (6.22) and the continuity of $\nabla\Phi_\tau$.

Next, it follows from (6.28) that the sequence $\{(A_{k_j+1}, E_{k_j+1}, Y_{k_j+1})\}$ also converges to (A^*, E^*, Y^*). Using this fact and taking limits of both sides of the equation

$$Y_{k_j+1} = Y_{k_j} + \rho(X - A_{k_j+1} - E_{k_j+1}),$$

which itself comes from line 5 of Algorithm 8, gives us (6.32).

Finally, we show that (6.31) holds. For any k_j, the update in line 3 of Algorithm 8 gives the following optimality condition:

$$\exists\ g_{k_j+1} \in \gamma\partial\|E_{k_j+1}\|_1 \ \text{ such that } \ g_{k_j+1} - Y_{k_j} - \mu(X - A_{k_j} - E_{k_j+1}) = 0. \tag{6.33}$$

Also, it follows from convexity of $\|\cdot\|_1$ that

$$\gamma\|E\|_1 - \gamma\|E_{k_j+1}\|_1 \geq \left\langle g_{k_j+1}, E - E_{k_j+1}\right\rangle \quad \text{for all } E.$$

Combing this inequality with (6.33) shows that

$$\gamma\|E\|_1 - \gamma\|E_{k_j+1}\|_1 \geq \left\langle Y_{k_j} + \mu(X - A_{k_j} - E_{k_j+1}), E - E_{k_j+1}\right\rangle \quad \text{for all } E.$$

Taking limits on both sides and using (6.32), which we already proved, gives

$$\gamma\|E\|_1 - \gamma\|E^*\|_1 \geq \langle Y^*, E - E^*\rangle \quad \text{for all } E.$$

Rearranging the terms, we see that

$$\gamma\|E\|_1 - \langle Y^*, E\rangle \geq \gamma\|E^*\|_1 - \langle Y^*, E^*\rangle \quad \text{for all } E,$$

which means that the matrix E^* satisfies

$$E^* = \underset{E}{\operatorname{argmin}}\, \gamma\|E\|_1 - \langle Y^*, E\rangle. \tag{6.34}$$

The proof is complete once we observe that E^* must satisfy the first-order optimality condition for problem (6.34), which is precisely (6.31). □

6.6 Numerical experiments

In this section, we discuss our implementation of LRSC-ADMM and evaluate its performance when applied to subspace clustering problems. The measure of performance we use is the subspace clustering accuracy which is defined as

$$\text{subspace clustering accuracy} = \frac{\text{number of correctly classified points}}{\text{total number of points}}.$$

We compare LRSC-ADMM to three other low-rank based subspace clustering methods: LRR [72, 73], REDU-EXPR [75], and a method based on solving problem (6.6). LRR was originally designed to do subspace clustering for data corrupted by outliers [72]. In [73], the authors briefly mentioned an ℓ_1 variant of the original LRR that handles data corrupted by sparse errors instead of outliers. It is this ℓ_1 variant (see (6.36)) that we implemented in our numerical experiments. It should be pointed out that our implementations of LRR and REDU-EXPR are designed to do subspace clustering for data that only contains sparse gross errors, while the other two methods are designed for problems with both dense noise and sparse errors. Nonetheless, we compare these four methods to see if we gain anything by solving more complex optimization problems that account for both types of noise. In the next few paragraphs, we provide more details about the three methods with which LRSC-ADMM is compared.

LRR is one of the earliest low-rank based subspace clustering methods. It was

originally designed to handle data corrupted by outliers. Given a $d \times n$ data matrix X which contains outliers, LRR obtains an affinity matrix by solving the convex problem

$$\underset{C,E}{\text{minimize}} \ \|C\|_* + \gamma \|E\|_{2,1} \ \text{ subject to } \ X = XC + E, \tag{6.35}$$

where $\|E\|_{2,1} = \sum_{j=1}^{n} \sqrt{\sum_{i=1}^{n} (E_{ij})^2}$ is the $\ell_{2,1}$ norm and $\gamma > 0$ is a weighting parameter that needs to be tuned in practice.

In our numerical experiments, since our data contains sparse errors instead of outliers, we use a variant of (6.35) that replaces the $\ell_{2,1}$ norm by an ℓ_1 norm and as a result solve the following convex optimization problem

$$\underset{C,E}{\text{minimize}} \ \|C\|_* + \gamma \|E\|_1 \ \text{ subject to } \ X = XC + E, \tag{6.36}$$

which was mentioned in [73] and [65].

After getting the optimal C^*, we set the affinity matrix to be $|C^*| + |C^{*T}|$ and use it to perform spectral clustering. Basically, this method uses the corrupted matrix X itself as a dictionary and tries to get a low rank representation C for the purpose of spectral clustering. The problem (6.36) is convex and can be solved by ADMM.

REDU-EXPR computes an affinity matrix by the following two-step procedure. Given a noisy matrix X, it first tries to remove the noise and obtain a "clean" data

matrix B^* by performing robust PCA on X:

$$\underset{B,E}{\text{minimize}} \ \|B\|_* + \gamma\|E\|_1 \ \text{ subject to } \ X = B + E, \tag{6.37}$$

where $\gamma > 0$ is a parameter to be tuned. After getting a minimizer B^* to the problem above, REDU-EXPR computes a matrix C^* using the result in Section 6.2, i.e., $C^* = V_1 V_1^T$, where V_1 contains the right singular vectors from the compact SVD of B^* (see Lemma 6.2.1). The affinity matrix is then set to be $|C^*|$ and spectral clustering is applied using this affinity matrix. The problem (6.37) is convex and can be solved using ADMM.

We may see that LRR and REDU-EXPR make a lot of sense when the data is only contaminated by sparse errors. When there is only dense noise, we may change the norm $\|\cdot\|_1$ in (6.36) and (6.37) to the Frobenius norm squared and still expect decent performance from these two methods. However, when there are both types of noise, we do not have a good reason to expect them to perform well. The third method which we now describe has the same spirit as REDU-EXPR and is designed for problems with both dense noise and sparse errors.

The method is based on problem (6.6). By Lemma 6.2.1, solving (6.6) is equivalent to solving

$$\begin{aligned} &\underset{B,E,G}{\text{minimize}} \ \ \text{rank}(B) + \frac{\alpha}{2}\|G\|_F^2 + \gamma\|E\|_1 \\ &\text{subject to} \ \ X = B + G + E, \end{aligned} \tag{6.38}$$

and computing the optimal C^* by $C^* = V_1 V_1^T$, where V_1 contains the right singular vectors from the compact SVD of the optimal B^* to problem (6.38). Problem (6.38) is difficult to solve and is therefore relaxed to

$$\begin{aligned} \underset{B,E,G}{\text{minimize}} \quad & \|B\|_* + \frac{\alpha}{2}\|G\|_F^2 + \gamma\|E\|_1 \\ \text{subject to} \quad & X = B + G + E. \end{aligned} \tag{6.39}$$

The whole method is to first solve the convex optimization problem (6.39) for B^*, E^* and G^*, second to compute C^* by using the computed B^* as described above, third to set $|C^*|$ as the affinity matrix, and fourth to apply spectral clustering. It turns out that this method is similar to REDU-EXPR with the main difference being that we model both types of noise with the optimization problem formulation. Therefore we refer to this method as REDU-EXPR-2 in the remainder of this chapter. To solve the convex problem (6.39), we transform it to an unconstrained problem by substituting $G = X - B - E$ (this follows from the constraint) in the objective and then applying an alternating minimization algorithm, which is guaranteed to converge [79].

We compared these subspace clustering methods on three problems that involve data with both sparse errors and small dense noise: two synthetic problems and a face clustering problem. Except for LRR, whose MATLAB codes were provided by the authors, we coded the algorithms in MATLAB ourselves. All experiments were run on a single core of an Intel Core i7-4510U CPU with 8 GB of RAM.

Before going into the details of our experimental setups, we want to remark on

two points of our implementation. The first is that we used $|C^*| + |C^*|^T$ instead of $|C^*|$ as the affinity matrix in spectral clustering to ensure that it is symmetric. The second is that we used the k-means clustering algorithm in the spectral clustering phase. Since the performance of k-means depended on its initialization, for each given affinity matrix, we ran 20 k-means trials with random initializations and returned the clustering result of the trial that gave the smallest k-means error based on squared Euclidean distance.

6.6.1 Results on synthetic data

In this section, we compare the performance of LRSC-ADMM to LRR, REDU-EXPR and REDU-EXPR-2 using two synthetic problems.

6.6.1.1 Synthetic problem #1

We randomly generate 5 independent subspaces in $\mathbb{R}^{30}$. For the i-th subspace, the dimension d_i is set to be i. So the 5 subspaces have dimensions 1 through 5. For each trial, we randomly sample $10d_i$ points from subspace i, where each point is generated by right multiplying U_i, an orthonormal basis for subspace i, with a d_i dimensional vector whose elements are independent Gaussian random variables with mean 0 and variance 1. This gives us a 30×150 clean data matrix whose largest element in magnitude is typically about 2. We then add two types of noise to the clean data matrix. First, for subspace i, we add zero-mean Gaussian noise with covariance

$0.01(I - U_i U_i^T)$, where U_i is again an orthonormal basis for the subspace. We then randomly select $\beta\%$ of the elements in the whole data matrix, and then add to each of these selected elements a value uniformly generated on $[-1, 1]$. Therefore, the data is corrupted by both small dense noise and sparse errors. Given the noisy data matrix $X \in \mathbb{R}^{30\times 150}$, we apply different subspace clustering methods and record the mean and median clustering accuracy as well as the average computational time for each method over 100 random trials. Experiments for the three values $\beta \in \{5, 10, 20\}$ are conducted.

For each value of β, we tuned the parameters in the optimization problems of each method except for REDU-EXPR. We set $\gamma = 1/\sqrt{\max\{d, n\}}$, ($d = 30$, $n = 150$ here) for REDU-EXPR as suggested in [75]. All the parameters are summarized in Table 6.1. In LRR, REDU-EXPR and LRSC-ADMM, ADMM is used to solve the corresponding optimization problems. The penalty parameters in ADMM were set in different ways for these three methods. For LRR, the penalty parameter was set as suggested in [73]. For REDU-EXPR, we set the penalty parameter to be $nd/(4\|X\|_1)$ as suggested in [80]. For LRSC-ADMM, recall that our convergence results require the penalty parameter to be larger than 6τ (see Assumption 6.5.1). In our experiments, setting it to be 7τ gave good empirical results.

Tables 6.2 shows the mean and median clustering accuracies and the average computational time for the four methods under three different levels of sparse errors, namely $\beta = 5$, $\beta = 10$, and $\beta = 20$.

Table 6.1: Parameters used by different methods on synthetic problem #1.

method	β	γ	τ	α
LRR	5	0.046		
	10	0.046		
	20	0.028		
REDU-EXPR	5	0.0816		
	10	0.0816		
	20	0.0816		
REDU-EXPR-2	5	0.13		0.6
	10	0.1		0.8
	20	0.1		0.6
LRSC-ADMM	5	0.03	0.8	
	10	0.04	0.3	
	20	0.03	0.2	

From Table 6.2, we see that the performance of all methods in terms of clustering accuracy tends to deteriorate when the level of sparse noise increases. In general, LRSC-ADMM can achieve clustering accuracies on par with or slightly better than those of LRR and REDU-EXPR. However, it does not perform as well as REDU-EXPR-2. These results show that introducing more complex optimization problems to model both dense noise and sparse errors can provide gains in terms of clustering accuracy. One possible reason that REDU-EXPR-2 achieves better accuracies than LRSC-ADMM is that Algorithm 8 is not guaranteed to find a global solution to (6.7). When we consider computational time, REDU-EXPR-2 is again the best out of the four methods and LRSC-ADMM is relatively slow compared to the other three. However, we should point out that the computational time of these methods largely depends on the data. As we will see in the experiments on synthetic problem #2

Table 6.2: Clustering accuracy (in percentage) and computational time (in seconds) for different algorithms on synthetic problem #1 with different levels of sparse errors.

sparse error level (β)	5	10	20
mean accuracy			
LRR	87.91	82.85	76.24
REDU-EXPR	87.49	83.99	77.30
REDU-EXPR-2	92.42	88.51	81.95
LRSC-ADMM	90.08	84.75	74.21
median accuracy			
LRR	87.33	81.33	76.00
REDU-EXPR	89.33	84.00	77.33
REDU-EXPR-2	94.67	89.33	82.00
LRSC-ADMM	90.67	83.33	74.00
mean time			
LRR	1.40	1.11	1.48
REDU-EXPR	0.32	0.31	0.30
REDU-EXPR-2	0.15	0.21	0.19
LRSC-ADMM	8.62	5.29	7.53

and the face clustering problem, it is not always the case that LRSC-ADMM is the slowest.

6.6.1.2 Synthetic problem #2

We randomly generate 6 independent 5-dimensional subspaces in $\mathbb{R}^{30}$. For each trial, we randomly sample 50 points from each subspace, where each point is generated by right multiplying U_i, an orthonormal basis for subspace i, with a 5-dimensional vector whose elements are independent Gaussian random variables with mean 0 and variance 1. This gives us a 30×300 clean data matrix whose largest element in magni-

tude is typically about 2. We then add the same two types of noise as in the previous section to the clean data matrix. First, for subspace i, we add zero-mean Gaussian noise with covariance $0.01(I - U_i U_i^T)$, where U_i is again an orthonormal basis for the subspace. We then randomly select $\beta\%$ of the elements in the whole data matrix, and then add to each of these selected elements a value uniformly generated on $[-1, 1]$. Therefore, the data is corrupted by both small dense noise and sparse errors. Given the noisy data matrix $X \in \mathbb{R}^{30\times 300}$, we apply different subspace clustering methods and record the mean and median clustering accuracy as well as the average computational time for each method over 100 random trials. Experiments for the three values $\beta \in \{5, 10, 20\}$ are conducted. Notice that the clean data matrix generated in this way has a rank of 30, which is equal to the dimension of the ambient space. Thus, this experiment tests the performance of each method in the case when the clean data matrix is not really low-rank compared to the dimension of the ambient space.

The weighting parameters used in different methods are listed in Table 6.3. The penalty parameters used in the ADMM algorithms in different methods are set in the same way as in the previous section.

Tables 6.4 shows the mean and median clustering accuracies and the average computational time for the four methods on synthetic problem #2.

We can observe from Table 6.4 that all four methods perform almost equally well in terms of clustering accuracy. LRSC-ADMM is slightly less accurate than the others when $\beta = 5$ or 10. However, unlike in the previous section, LRSC-ADMM is

Table 6.3: Parameters used by different methods on synthetic problem #2.

method	β	γ	τ	α
LRR	5	0.02		
	10	0.015		
	20	0.01		
REDU-EXPR	5	0.09		
	10	0.09		
	20	0.08		
REDU-EXPR-2	5	0.08		0.55
	10	0.08		0.7
	20	0.08		0.4
LRSC-ADMM	5	0.09	0.1	
	10	0.06	0.1	
	20	0.095	0.05	

now the fastest method for this particular set of experiments. REDU-EXPR-2 has once again the best overall performance when we consider both clustering accuracy and computational time. It is also worth noting that all methods perform better than in the previous section in terms of clustering accuracy. This suggests that subspace clustering might be easier when the true subspaces have similar dimensions and contain a similar amount of data points, which is also hinted at by the numerical results presented in [65, Chapter 8].

6.6.2 Results on face clustering

Face clustering is the problem of clustering a set of face images from multiple individuals according to the identity of each individual. For this problem, each data point (i.e. image) is represented by a long column vector that is the concatenation

Table 6.4: Clustering accuracy (in percentage) and computational time (in seconds) for different algorithms on synthetic problem #2 with different levels of sparse errors.

sparse error level (β)	5	10	20
mean accuracy			
LRR	99.33	98.62	95.21
REDU-EXPR	99.04	98.07	94.36
REDU-EXPR-2	99.27	98.40	95.31
LRSC-ADMM	99.12	97.81	93.47
median accuracy			
LRR	99.33	98.67	95.33
REDU-EXPR	99.00	98.00	94.33
REDU-EXPR-2	99.33	98.33	95.33
LRSC-ADMM	99.17	98.00	93.67
mean time			
LRR	4.63	5.00	5.37
REDU-EXPR	0.34	0.36	0.42
REDU-EXPR-2	0.29	0.38	0.28
LRSC-ADMM	0.22	0.24	0.25

of all the columns of the original image matrix whose elements are between 0 and 1. Therefore the length of the data point is equal to the number of pixels of the image and as a result, each data point lies in a high dimensional ambient space. For a Lambertian object, the set of all images taken under the same viewpoint and expression but different lighting conditions lie approximately in a low dimensional subspace [81]. Therefore, the face clustering problem can be treated as a subspace clustering problem. In practice, due to cast shadows and specularities, a few pixels of the face image can have large errors. Therefore, we have a clustering problem of data corrupted by small noise and sparse gross errors. (Small noise can be understood

from the fact that face images do not lie perfectly in a low dimensional subspace.)

We use the Extended Yale B database [82] to evaluate the performance of LRR, REDU-EXPR, REDU-EXPR-2, and LRSC-ADMM. The database includes 64 frontal face images of 38 individuals acquired under 64 different lighting conditions. To reduce the computational cost of all algorithms, we downsample the original images of 192 by 168 pixels to 48 by 42 pixels and treat each 2016-dimensional vectorized image as a data point. Some sample images from the database are shown in Figure 6.1.

Figure 6.1: Sample images of the faces of three subjects under three different illumination conditions.

We apply the four methods to cluster N subjects where $N \in \{2, 10, 20, 38\}$. For $N \in \{2, 10, 20\}$, we record the mean and median clustering accuracies and the average computational time over 20 trials where in each trial, we randomly select N subjects out of the 38. We also record the clustering accuracy and computational time when we apply the methods to the whole dataset of 38 subjects.

For each method, we tuned the parameters using $N = 10$ subjects and applied the same parameters when we cluster $N \in \{2, 20, 38\}$ subjects. The parameters we used are listed in Table 6.5. When we applied ADMM in LRR, REDU-EXPR, and LRSC-ADMM, we set the penalty parameters in the same way as we did in the last section. Table 6.6 shows the results of our experiments.

Table 6.5: Parameters used in different methods for the face clustering problem

method	γ	τ	α
LRR	0.003		
REDU-EXPR	0.01		
REDU-EXPR-2	0.02		0.5
LRSC-ADMM	0.02	0.05	

As we can see from Table 6.6, no method is dominant in terms of clustering accuracy. When the number of subjects is 2, LRR and LRSC-ADMM perform better than the other two methods. But as the number of subjects increases, REDU-EXPR and REDU-EXPR-2 tend to give higher clustering accuracies. The reason why REDU-EXPR-2 does not show a dominant performance as in the last section is partly due to the fact that we only tuned the methods using 10 subjects instead of tuning them for all cases. It is also interesting to see that for this set of experiments, LRSC-ADMM requires less computational time than the other three methods except in the case when the number of subjects is 2. Together with the experimental results on synthetic problems, we find it hard to draw a conclusion about the computational efficiency of the four methods. We also remark that parameter tuning is crucial to the performance of these methods. For example, for LRSC-ADMM, it is possible that with a change of parameter values, we get slightly better clustering accuracies with a sacrifice in efficiency. To be precise, if we set $\tau = 0.06$ and $\gamma = 0.003$, we get the results in Table 6.7. Therefore, when we apply these methods in practice, some effort in parameter tuning is necessary.

To summarize, we feel that with parameter tuning, REDU-EXPR-2 gives the best

Table 6.6: Clustering accuracy (in percentage) and computational time (in seconds) for the different algorithms on the Extended Yale B database.

No. subjects	2	10	20	38
mean accuracy				
LRR	97.75	68.78	69.71	68.72
REDU-EXPR	91.94	73.44	73.14	71.38
REDU-EXPR-2	90.86	73.45	71.25	76.10
LRSC-ADMM	97.47	66.61	71.21	70.84
median accuracy				
LRR	98.44	70.13	69.63	68.72
REDU-EXPR	93.23	71.25	74.87	71.38
REDU-EXPR-2	92.97	74.02	71.16	76.10
LRSC-ADMM	97.21	63.74	71.42	70.84
mean time				
LRR	5.03	87.44	628.95	3361.94
REDU-EXPR	16.13	334.93	1566.69	4595.76
REDU-EXPR-2	2.78	36.15	264.52	789.36
LRSC-ADMM	69.37	34.10	142.00	292.06

overall performance in terms of clustering accuracy and computational cost for subspace clustering problems with both dense noise and sparse errors. The performance of LRSC-ADMM is at least on par with those of LRR and REDU-EXPR and not too far behind that of REDU-EXPR-2.

6.7 Conclusion

In this chapter, we looked at the subspace clustering problem and revisited an LRSC method (6.7) proposed in [74], which requires the solution to nonconvex structured optimization problems. We showed how to compute a first-order solution to this

Table 6.7: Clustering accuracy (in percentage) and computational time (in seconds) of LRSC-ADMM on the Extended Yale B database with $\tau = 0.06$, $\gamma = 0.003$.

No. subjects	2	10	20	38
mean accuracy	97.88	70.30	70.19	69.01
median accuray	98.41	71.96	70.52	69.01
mean time	567.13	401.35	1718.81	4071.61

nonconvex problem by using an ADMM scheme. Experiments on synthetic and real data illustrated that the LRSC model together with ADMM could achieve clustering accuracies on par with state-of-the-art methods for subspace clustering problems with noisy data. Overall, solving another model formulation (6.6) using the way we discussed in §6.6 performed the best.

Bibliography

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

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

[3] A. Barclay, P. E. Gill, and J. B. Rosen, "SQP methods and their application to numerical optimal control," in *Variational calculus, optimal control and applications*, pp. 207–222, Springer, 1998.

[4] P. E. Gill, L. O. Jay, M. W. Leonard, L. R. Petzold, and V. Sharma, "An SQP method for the optimal control of large-scale dynamical systems," *Journal of computational and applied mathematics*, vol. 120, no. 1, pp. 197–213, 2000.

[5] D. Kraft, "On converting optimal control problems into nonlinear programming problems," in *Computational mathematical programming*, pp. 261–280, Springer, 1985.

[6] G. Alarcón, C. Torres, and L. Gómez, "Global optimization of gas allocation to a group of wells in artificial lift using nonlinear constrained programming," *Journal of energy resources technology*, vol. 124, p. 262, 2002.

[7] T. Johansen, T. Fossen, and S. Berge, "Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 1, pp. 211–216, 2004.

[8] P. Liu and A. Der Kiureghian, "Optimization algorithms for structural reliability," *Structural safety*, vol. 9, no. 3, pp. 161–177, 1991.

[9] M. Bazaraa, H. Sherali, and C. Shetty, *Nonlinear programming: theory and algorithms*. John Wiley and Sons, 2006.

[10] S. Sra, S. Nowozin, and S. J. Wright, *Optimization for machine learning*. MIT Press, 2012.

[11] S. Bubeck *et al.*, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[12] D. Fernández and M. V. Solodov, "Local convergence of exact and inexact augmented lagrangian methods under the second-order sufficient optimality condition," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 384–407, 2012.

[13] A. F. Izmailov and M. V. Solodov, "On attraction of linearly constrained la-

grangian methods and of stabilized and quasi-newton SQP methods to critical multipliers," *Mathematical programming*, vol. 126, no. 2, pp. 231–257, 2011.

[14] R. H. Byrd, G. Lopez-Calva, and J. Nocedal, "A Line Search Exact Penalty Method Using Steering Rules," *Mathematical Programming*, vol. 133, no. 1–2, pp. 39–73, 2012.

[15] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Steering exact penalty methods for nonlinear programming," *Optimization Methods and Software*, vol. 23, no. 2, pp. 197–213, 2008.

[16] M. Mongeau and A. Sartenaer, "Automatic decrease of the penalty parameter in exact penalty function methods," *European Journal of Operational Research*, vol. 83, no. 3, pp. 686–699, 1995.

[17] A. R. Conn, N. I. M. Gould, and P. L. Toint, `LANCELOT`*: a Fortran package for large-scale nonlinear optimization (Release A).* Lecture Notes in Computation Mathematics 17, Springer Verlag, 1992.

[18] N. I. Gould, D. Orban, and P. L. Toint, "CUTEr and SifDec: A constrained and unconstrained testing environment, revisited," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 4, pp. 373–394, 2003.

[19] A. Bondarenko, D. Bortz, and J. J. Moré, "COPS: Large-scale nonlinearly constrained optimization problems," Technical Report ANL/MCS-TM-237, Mathe-

matics and Computer Science division, Argonne National Laboratory, Argonne, IL, 1998. Revised October 1999.

[20] 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," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[21] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.

[22] M. J. D. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization* (R. Fletcher, ed.), pp. 283–298, Academic Press, 1969.

[23] 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*, vol. 111, no. 1-2, pp. 5–32, 2008.

[24] E. G. Birgin and J. M. Martínez, "Augmented lagrangian method with nonmonotone penalty parameters for constrained optimization," *Computational Optimization and Applications*, vol. 51, no. 3, pp. 941–965, 2012.

[25] A. R. Conn, N. I. Gould, and P. Toint, "A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

[26] 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)*, vol. 82 of *Appl. Optim.*, pp. 303–321, Norwell, MA: Kluwer Acad. Publ., 2003.

[27] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires," *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, vol. 9, no. 2, pp. 41–76, 1975.

[28] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[29] J. Eckstein, *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, Massachusetts Institute of Technology, 1989.

[30] J. Eckstein and D. P. Bertsekas, "On the douglasrachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1-3, pp. 293–318, 1992.

[31] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[32] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *Journal of the ACM (JACM)*, vol. 58, no. 3, p. 11, 2011.

[33] E. Elhamifar and R. Vidal, "Sparse subspace clustering: Algorithm, theory, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2765–2781, 2013.

[34] K. Scheinberg, S. Ma, and D. Goldfarb, "Sparse inverse covariance selection via alternating linearization methods," in *Advances in neural information processing systems*, pp. 2101–2109, 2010.

[35] M. A. Figueiredo and J. M. Bioucas-Dias, "Restoration of poissonian images using alternating direction optimization," *IEEE Transactions on Image Processing*, vol. 19, no. 12, pp. 3133–3145, 2010.

[36] P. L. Combettes and J.-C. Pesquet, "A douglas–rachford splitting approach to nonsmooth convex variational signal recovery," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 564–574, 2007.

[37] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in ad hoc wsns with noisy linkspart i: Distributed estimation of deterministic signals," *IEEE Transactions on Signal Processing*, vol. 56, no. 1, pp. 350–364, 2008.

[38] I. D. Schizas, G. B. Giannakis, S. I. Roumeliotis, and A. Ribeiro, "Consensus in ad hoc wsns with noisy linkspart ii: Distributed estimation and smoothing

of random signals," *IEEE Transactions on Signal Processing*, vol. 56, no. 4, pp. 1650–1666, 2008.

[39] C. Feng, H. Xu, and B. Li, "An alternating direction method approach to cloud traffic management," *arXiv preprint arXiv:1407.8309*, 2014.

[40] W.-C. Liao, M. Hong, H. Farmanbar, X. Li, Z.-Q. Luo, and H. Zhang, "Min flow rate maximization for software defined radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1282–1294, 2014.

[41] D. P. Bertsekas, *Convex optimization algorithms*. Athena Scientific, 2015.

[42] 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*, vol. 3, no. 1, pp. 1–122, 2011.

[43] J.-J. Moreau, "Fonctions convexes duales et points proximaux dans un espace hilbertien," *CR Acad. Sci. Paris Sér. A Math*, vol. 255, pp. 2897–2899, 1962.

[44] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-point algorithms for inverse problems in science and engineering*, pp. 185–212, Springer, 2011.

[45] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *arXiv preprint arXiv:1410.1390*, 2014.

[46] G. Li and T. K. Pong, "Global convergence of splitting methods for nonconvex composite optimization," *SIAM Journal on Optimization*, vol. 25, no. 4, pp. 2434–2460, 2015.

[47] F. Wang, Z. Xu, and H.-K. Xu, "Convergence of bregman alternating direction method with multipliers for nonconvex composite problems," *arXiv preprint arXiv:1410.8625*, 2014.

[48] P. L. Toint, "Nonlinear stepsize control, trust regions and regularizations for unconstrained optimization," *Optimization Methods and Software*, vol. 28, no. 1, pp. 82–95, 2013.

[49] J. J. Moré, "Trust regions and projected gradients," in *System Modelling and Optimization* (M. Iri and K. Yajima, eds.), vol. 113 of *Lecture Notes in Control and Information Sciences*, pp. 1–13, Springer Berlin Heidelberg, 1988.

[50] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2000.

[51] K. R. Davidson and A. P. Donsig, *Real analysis and applications*. Undergraduate Texts in Mathematics, New York: Springer, 2010.

[52] N. I. M. Gould, , D. Orban, and Ph. L. Toint, "CUTEst : A constrained and unconstrained testing environment with safe threads for mathematical optimiza-

tion," *Computational Optimization and Applications*, vol. 60, no. 3, pp. 545–557, 2015.

[53] E. G. Birgin and J. M. Martínez, "Improving ultimate convergence of an augmented Lagrangian method," *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, 2008.

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

[55] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, Ser. A, pp. 201–213, 2002.

[56] J. L. Morales, "A numerical study of limited memory BFGS methods," *Applied Mathematics Letters*, vol. 15, no. 4, pp. 481–487, 2002.

[57] 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*, vol. 73, no. 1, pp. 73–110, 1996.

[58] 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*, vol. 29, no. 4, pp. 353–372, 2003.

[59] A. R. Conn, N. I. M. Gould, and P. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM journal on numerical analysis*, vol. 25, no. 2, pp. 433–460, 1988.

[60] 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*, vol. 50, no. 182, pp. 399–430, 1988.

[61] R. Vidal, R. Tron, and R. Hartley, "Multiframe motion segmentation with missing data using PowerFactorization and GPCA," *International Journal of Computer Vision*, vol. 79, no. 1, pp. 85–105, 2008.

[62] J. Ho, M.-H. Yang, J. Lim, K.-C. Lee, and D. Kriegman, "Clustering appearances of objects under varying illumination conditions," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. I–11, IEEE, 2003.

[63] W. Hong, J. Wright, K. Huang, and Y. Ma, "Multiscale hybrid linear models for lossy image representation," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3655–3671, 2006.

[64] R. Vidal, "Subspace clustering," *IEEE Signal Processing Magazine*, vol. 28, pp. 52–68, March 2011.

[65] R. Vidal, Y. Ma, and S. Sastry, *Generalized Principal Component Analysis*. Springer Interdisciplinary Applied Mathematics, 2016.

[66] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[67] E. Elhamifar and R. Vidal, "Sparse subspace clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2790–2797, 2009.

[68] E. Elhamifar and R. Vidal, "Clustering disjoint subspaces via sparse representation," in *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pp. 1926–1929, 2010.

[69] M. Soltanolkotabi and E. J. Candes, "A geometric analysis of subspace clustering with outliers," *The Annals of Statistics*, pp. 2195–2238, 2012.

[70] M. Soltanolkotabi, E. Elhamifar, and E. Candes, "Robust subspace clustering," *The Annals of Statistics*, vol. 42, no. 2, pp. 669–699, 2014.

[71] C. You and R. Vidal, "Geometric conditions for subspace-sparse recovery," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1585–1593, 2015.

[72] G. Liu, Z. Lin, and Y. Yu, "Robust subspace segmentation by low-rank representation.," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 663–670, 2010.

[73] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust recovery of subspace

structures by low-rank representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 171–184, 2013.

[74] R. Vidal and P. Favaro, "Low rank subspace clustering (LRSC)," *Pattern Recognition Letters*, vol. 43, pp. 47–61, 2014.

[75] H. Zhang, Z. Lin, C. Zhang, and J. Gao, "Relations among some low-rank subspace recovery models," *Neural computation*, 2015.

[76] W. Rudin, *Principles of mathematical analysis*, vol. 3. McGraw-Hill New York, 1964.

[77] A. S. Lewis and H. S. Sendov, "Nonsmooth analysis of singular values. part i: Theory," *Set-Valued Analysis*, vol. 13, no. 3, pp. 213–241, 2005.

[78] F. Andersson, M. Carlsson, and K.-M. Perfekt, "Operator-lipschitz estimates for the singular value functional calculus," *Proceedings of the American Mathematical Society*, 2015.

[79] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications*, vol. 109, no. 3, pp. 475–494, 2001.

[80] X. Yuan and J. Yang, "Sparse and low-rank matrix decomposition via alternating direction methods," *preprint*, 2009.

[81] R. Basri and D. W. Jacobs, "Lambertian reflectance and linear subspaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 218–233, 2003.

[82] K.-C. Lee, J. Ho, and D. J. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 5, pp. 684–698, 2005.

Vita

Hao Jiang was born on August 22, 1988 in Shanghai, China. He attended Shanghai High School and graduated in 2006. He received his Bachelor of Science degree in Mathematics with first-class honors from the University of Hong Kong in 2010. He also studied at the University of California, Berkeley for one semester as an exchange student in 2009. In August 2010, Hao enrolled in the Ph.D. program in the Department of Applied Mathematics and Statistics at the Johns Hopkins University. He received a Master of Science in Engineering degree from that department in 2012.