Foundations and Trends® in
Machine Learning
Vol. 3, No. 1 (2010) 1–122
© 2011 S. Boyd, N. Parikh, E. Chu, B. Peleato
and J. Eckstein
DOI: 10.1561/2200000016

now
the essence of knowledge

Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers

**Stephen Boyd[1], Neal Parikh[2], Eric Chu[3]
Borja Peleato[4] and Jonathan Eckstein[5]**

[1] *Electrical Engineering Department, Stanford University, Stanford, CA 94305, USA, boyd@stanford.edu*
[2] *Computer Science Department, Stanford University, Stanford, CA 94305, USA, npparikh@cs.stanford.edu*
[3] *Electrical Engineering Department, Stanford University, Stanford, CA 94305, USA, echu508@stanford.edu*
[4] *Electrical Engineering Department, Stanford University, Stanford, CA 94305, USA, peleato@stanford.edu*
[5] *Management Science and Information Systems Department and RUTCOR, Rutgers University, Piscataway, NJ 08854, USA, jeckstei@rci.rutgers.edu*

Contents

1 Introduction **3**

2 Precursors **7**

2.1 Dual Ascent 7
2.2 Dual Decomposition 9
2.3 Augmented Lagrangians and the Method of Multipliers 10

3 Alternating Direction Method of Multipliers **13**

3.1 Algorithm 13
3.2 Convergence 15
3.3 Optimality Conditions and Stopping Criterion 18
3.4 Extensions and Variations 20
3.5 Notes and References 23

4 General Patterns **25**

4.1 Proximity Operator 25
4.2 Quadratic Objective Terms 26
4.3 Smooth Objective Terms 30
4.4 Decomposition 31

5 Constrained Convex Optimization **33**

5.1 Convex Feasibility 34
5.2 Linear and Quadratic Programming 36

6 ℓ_1-Norm Problems **38**

6.1 Least Absolute Deviations 39
6.2 Basis Pursuit 41
6.3 General ℓ_1 Regularized Loss Minimization 42
6.4 Lasso 43
6.5 Sparse Inverse Covariance Selection 45

7 Consensus and Sharing **48**

7.1 Global Variable Consensus Optimization 48
7.2 General Form Consensus Optimization 53
7.3 Sharing 56

8 Distributed Model Fitting **61**

8.1 Examples 62
8.2 Splitting across Examples 64
8.3 Splitting across Features 66

9 Nonconvex Problems **73**

9.1 Nonconvex Constraints 73
9.2 Bi-convex Problems 76

10 Implementation **78**

10.1 Abstract Implementation 78
10.2 MPI 80
10.3 Graph Computing Frameworks 81
10.4 MapReduce 82

11 Numerical Examples **87**

11.1 Small Dense Lasso 88
11.2 Distributed ℓ_1 Regularized Logistic Regression 92
11.3 Group Lasso with Feature Splitting 95
11.4 Distributed Large-Scale Lasso with MPI 97
11.5 Regressor Selection 100

12 Conclusions **103**

Acknowledgments **105**

A Convergence Proof **106**

References **111**

Abstract

Many problems of recent interest in statistics and machine learning can be posed in the framework of convex optimization. Due to the explosion in size and complexity of modern datasets, it is increasingly important to be able to solve problems with a very large number of features or training examples. As a result, both the decentralized collection or storage of these datasets as well as accompanying distributed solution methods are either necessary or at least highly desirable. In this review, we argue that the *alternating direction method of multipliers* is well suited to distributed convex optimization, and in particular to large-scale problems arising in statistics, machine learning, and related areas. The method was developed in the 1970s, with roots in the 1950s, and is equivalent or closely related to many other algorithms, such as dual decomposition, the method of multipliers, Douglas–Rachford splitting, Spingarn's method of partial inverses, Dykstra's alternating projections, Bregman iterative algorithms for ℓ_1 problems, proximal methods, and others. After briefly surveying the theory and history of the algorithm, we discuss applications to a wide variety of statistical and machine learning problems of recent interest, including the lasso, sparse logistic regression, basis pursuit, covariance selection, support vector machines, and many others. We also discuss general distributed optimization, extensions to the nonconvex setting, and efficient implementation, including some details on distributed MPI and Hadoop MapReduce implementations.

1

Introduction

In all applied fields, it is now commonplace to attack problems through data analysis, particularly through the use of statistical and machine learning algorithms on what are often large datasets. In industry, this trend has been referred to as 'Big Data', and it has had a significant impact in areas as varied as artificial intelligence, internet applications, computational biology, medicine, finance, marketing, journalism, network analysis, and logistics.

Though these problems arise in diverse application domains, they share some key characteristics. First, the datasets are often extremely large, consisting of hundreds of millions or billions of training examples; second, the data is often very high-dimensional, because it is now possible to measure and store very detailed information about each example; and third, because of the large scale of many applications, the data is often stored or even collected in a distributed manner. As a result, it has become of central importance to develop algorithms that are both rich enough to capture the complexity of modern data, and scalable enough to process huge datasets in a parallelized or fully decentralized fashion. Indeed, some researchers [92] have suggested that even highly complex and structured problems may succumb most easily to relatively simple models trained on vast datasets.

Many such problems can be posed in the framework of convex optimization. Given the significant work on decomposition methods and decentralized algorithms in the optimization community, it is natural to look to parallel optimization algorithms as a mechanism for solving large-scale statistical tasks. This approach also has the benefit that one algorithm could be flexible enough to solve many problems.

This review discusses the alternating direction method of multipliers (ADMM), a simple but powerful algorithm that is well suited to distributed convex optimization, and in particular to problems arising in applied statistics and machine learning. It takes the form of a *decomposition-coordination* procedure, in which the solutions to small local subproblems are coordinated to find a solution to a large global problem. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization, two earlier approaches that we review in §2. It turns out to be equivalent or closely related to many other algorithms as well, such as Douglas-Rachford splitting from numerical analysis, Spingarn's method of partial inverses, Dykstra's alternating projections method, Bregman iterative algorithms for ℓ_1 problems in signal processing, proximal methods, and many others. The fact that it has been re-invented in different fields over the decades underscores the intuitive appeal of the approach.

It is worth emphasizing that the algorithm itself is not new, and that we do not present any new theoretical results. It was first introduced in the mid-1970s by Gabay, Mercier, Glowinski, and Marrocco, though similar ideas emerged as early as the mid-1950s. The algorithm was studied throughout the 1980s, and by the mid-1990s, almost all of the theoretical results mentioned here had been established. The fact that ADMM was developed so far in advance of the ready availability of large-scale distributed computing systems and massive optimization problems may account for why it is not as widely known today as we believe it should be.

The main contributions of this review can be summarized as follows:

(1) We provide a simple, cohesive discussion of the extensive literature in a way that emphasizes and unifies the aspects of primary importance in applications.

(2) We show, through a number of examples, that the algorithm is well suited for a wide variety of large-scale distributed modern problems. We derive methods for decomposing a wide class of statistical problems by training examples and by features, which is not easily accomplished in general.

(3) We place a greater emphasis on practical large-scale implementation than most previous references. In particular, we discuss the implementation of the algorithm in cloud computing environments using standard frameworks and provide easily readable implementations of many of our examples.

Throughout, the focus is on applications rather than theory, and a main goal is to provide the reader with a kind of 'toolbox' that can be applied in many situations to derive and implement a distributed algorithm of practical use. Though the focus here is on parallelism, the algorithm can also be used serially, and it is interesting to note that with no tuning, ADMM can be competitive with the best known methods for some problems.

While we have emphasized applications that can be concisely explained, the algorithm would also be a natural fit for more complicated problems in areas like graphical models. In addition, though our focus is on statistical learning problems, the algorithm is readily applicable in many other cases, such as in engineering design, multi-period portfolio optimization, time series analysis, network flow, or scheduling.

Outline

We begin in §2 with a brief review of dual decomposition and the method of multipliers, two important precursors to ADMM. This section is intended mainly for background and can be skimmed. In §3, we present ADMM, including a basic convergence theorem, some variations on the basic version that are useful in practice, and a survey of some of the key literature. A complete convergence proof is given in appendix A.

In §4, we describe some general patterns that arise in applications of the algorithm, such as cases when one of the steps in ADMM can

be carried out particularly efficiently. These general patterns will recur throughout our examples. In §5, we consider the use of ADMM for some generic convex optimization problems, such as constrained minimization and linear and quadratic programming. In §6, we discuss a wide variety of problems involving the ℓ_1 norm. It turns out that ADMM yields methods for these problems that are related to many state-of-the-art algorithms. This section also clarifies why ADMM is particularly well suited to machine learning problems.

In §7, we present *consensus* and *sharing* problems, which provide general frameworks for distributed optimization. In §8, we consider distributed methods for generic model fitting problems, including regularized regression models like the lasso and classification models like support vector machines.

In §9, we consider the use of ADMM as a heuristic for solving some nonconvex problems. In §10, we discuss some practical implementation details, including how to implement the algorithm in frameworks suitable for cloud computing applications. Finally, in §11, we present the details of some numerical experiments.

2

Precursors

In this section, we briefly review two optimization algorithms that are precursors to the alternating direction method of multipliers. While we will not use this material in the sequel, it provides some useful background and motivation.

2.1 Dual Ascent

Consider the equality-constrained convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b, \end{array} \tag{2.1}$$

with variable $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m \times n}$ and $f : \mathbf{R}^n \to \mathbf{R}$ is convex.

The Lagrangian for problem (2.1) is

$$L(x, y) = f(x) + y^T(Ax - b)$$

and the dual function is

$$g(y) = \inf_x L(x, y) = -f^*(-A^T y) - b^T y,$$

where y is the dual variable or Lagrange multiplier, and f^* is the convex conjugate of f; see [20, §3.3] or [140, §12] for background. The dual

problem is

$$\text{maximize} \quad g(y),$$

with variable $y \in \mathbf{R}^m$. Assuming that strong duality holds, the optimal values of the primal and dual problems are the same. We can recover a primal optimal point $x^\star$ from a dual optimal point $y^\star$ as

$$x^\star = \operatorname*{argmin}_x L(x, y^\star),$$

provided there is only one minimizer of $L(x, y^\star)$. (This is the case if, *e.g.*, f is strictly convex.) In the sequel, we will use the notation $\operatorname{argmin}_x F(x)$ to denote *any* minimizer of F, even when F does not have a unique minimizer.

In the *dual ascent method*, we solve the dual problem using gradient ascent. Assuming that g is differentiable, the gradient $\nabla g(y)$ can be evaluated as follows. We first find $x^+ = \operatorname{argmin}_x L(x, y)$; then we have $\nabla g(y) = Ax^+ - b$, which is the residual for the equality constraint. The dual ascent method consists of iterating the updates

$$x^{k+1} := \operatorname*{argmin}_x L(x, y^k) \tag{2.2}$$

$$y^{k+1} := y^k + \alpha^k (Ax^{k+1} - b), \tag{2.3}$$

where $\alpha^k > 0$ is a step size, and the superscript is the iteration counter. The first step (2.2) is an x-minimization step, and the second step (2.3) is a dual variable update. The dual variable y can be interpreted as a vector of prices, and the y-update is then called a *price update* or *price adjustment* step. This algorithm is called dual ascent since, with appropriate choice of α^k, the dual function increases in each step, *i.e.*, $g(y^{k+1}) > g(y^k)$.

The dual ascent method can be used even in some cases when g is not differentiable. In this case, the residual $Ax^{k+1} - b$ is not the gradient of g, but the negative of a *subgradient* of $-g$. This case requires a different choice of the α^k than when g is differentiable, and convergence is not monotone; it is often the case that $g(y^{k+1}) \not> g(y^k)$. In this case, the algorithm is usually called the *dual subgradient method* [152].

If α^k is chosen appropriately and several other assumptions hold, then x^k converges to an optimal point and y^k converges to an optimal

dual point. However, these assumptions do not hold in many applications, so dual ascent often cannot be used. As an example, if f is a nonzero affine function of any component of x, then the x-update (2.2) fails, since L is unbounded below in x for most y.

2.2 Dual Decomposition

The major benefit of the dual ascent method is that it can lead to a decentralized algorithm in some cases. Suppose, for example, that the objective f is *separable* (with respect to a partition or splitting of the variable into subvectors), meaning that

$$f(x) = \sum_{i=1}^{N} f_i(x_i),$$

where $x = (x_1, \ldots, x_N)$ and the variables $x_i \in \mathbf{R}^{n_i}$ are subvectors of x. Partitioning the matrix A conformably as

$$A = [A_1 \ \cdots \ A_N],$$

so $Ax = \sum_{i=1}^{N} A_i x_i$, the Lagrangian can be written as

$$L(x, y) = \sum_{i=1}^{N} L_i(x_i, y) = \sum_{i=1}^{N} \left(f_i(x_i) + y^T A_i x_i - (1/N) y^T b \right),$$

which is also separable in x. This means that the x-minimization step (2.2) splits into N separate problems that can be solved in parallel. Explicitly, the algorithm is

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} L_i(x_i, y^k) \tag{2.4}$$

$$y^{k+1} := y^k + \alpha^k (A x^{k+1} - b). \tag{2.5}$$

The x-minimization step (2.4) is carried out independently, in parallel, for each $i = 1, \ldots, N$. In this case, we refer to the dual ascent method as *dual decomposition*.

In the general case, each iteration of the dual decomposition method requires a *broadcast* and a *gather* operation. In the dual update step (2.5), the equality constraint residual contributions $A_i x_i^{k+1}$ are

collected (gathered) in order to compute the residual $Ax^{k+1} - b$. Once the (global) dual variable y^{k+1} is computed, it must be distributed (broadcast) to the processors that carry out the N individual x_i minimization steps (2.4).

Dual decomposition is an old idea in optimization, and traces back at least to the early 1960s. Related ideas appear in well known work by Dantzig and Wolfe [44] and Benders [13] on large-scale linear programming, as well as in Dantzig's seminal book [43]. The general idea of dual decomposition appears to be originally due to Everett [69], and is explored in many early references [107, 84, 117, 14]. The use of nondifferentiable optimization, such as the subgradient method, to solve the dual problem is discussed by Shor [152]. Good references on dual methods and decomposition include the book by Bertsekas [16, chapter 6] and the survey by Nedić and Ozdaglar [131] on distributed optimization, which discusses dual decomposition methods and consensus problems. A number of papers also discuss variants on standard dual decomposition, such as [129].

More generally, decentralized optimization has been an active topic of research since the 1980s. For instance, Tsitsiklis and his co-authors worked on a number of decentralized detection and consensus problems involving the minimization of a smooth function f known to multiple agents [160, 161, 17]. Some good reference books on parallel optimization include those by Bertsekas and Tsitsiklis [17] and Censor and Zenios [31]. There has also been some recent work on problems where each agent has its own convex, potentially nondifferentiable, objective function [130]. See [54] for a recent discussion of distributed methods for graph-structured optimization problems.

2.3 Augmented Lagrangians and the Method of Multipliers

Augmented Lagrangian methods were developed in part to bring robustness to the dual ascent method, and in particular, to yield convergence without assumptions like strict convexity or finiteness of f. The *augmented Lagrangian* for (2.1) is

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2, \tag{2.6}$$

where $\rho > 0$ is called the *penalty parameter*. (Note that L_0 is the standard Lagrangian for the problem.) The augmented Lagrangian can be viewed as the (unaugmented) Lagrangian associated with the problem

$$\begin{array}{ll} \text{minimize} & f(x) + (\rho/2)\|Ax - b\|_2^2 \\ \text{subject to} & Ax = b. \end{array}$$

This problem is clearly equivalent to the original problem (2.1), since for any feasible x the term added to the objective is zero. The associated dual function is $g_\rho(y) = \inf_x L_\rho(x, y)$.

The benefit of including the penalty term is that g_ρ can be shown to be differentiable under rather mild conditions on the original problem. The gradient of the augmented dual function is found the same way as with the ordinary Lagrangian, *i.e.*, by minimizing over x, and then evaluating the resulting equality constraint residual. Applying dual ascent to the modified problem yields the algorithm

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, y^k) \tag{2.7}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} - b), \tag{2.8}$$

which is known as the *method of multipliers* for solving (2.1). This is the same as standard dual ascent, except that the x-minimization step uses the augmented Lagrangian, and the penalty parameter ρ is used as the step size α^k. The method of multipliers converges under far more general conditions than dual ascent, including cases when f takes on the value $+\infty$ or is not strictly convex.

It is easy to motivate the choice of the particular step size ρ in the dual update (2.8). For simplicity, we assume here that f is differentiable, though this is not required for the algorithm to work. The optimality conditions for (2.1) are primal and dual feasibility, *i.e.*,

$$Ax^\star - b = 0, \qquad \nabla f(x^\star) + A^T y^\star = 0,$$

respectively. By definition, x^{k+1} minimizes $L_\rho(x, y^k)$, so

$$\begin{aligned} 0 &= \nabla_x L_\rho(x^{k+1}, y^k) \\ &= \nabla_x f(x^{k+1}) + A^T \Big(y^k + \rho(Ax^{k+1} - b) \Big) \\ &= \nabla_x f(x^{k+1}) + A^T y^{k+1}. \end{aligned}$$

We see that by using ρ as the step size in the dual update, the iterate (x^{k+1}, y^{k+1}) is dual feasible. As the method of multipliers proceeds, the primal residual $Ax^{k+1} - b$ converges to zero, yielding optimality.

The greatly improved convergence properties of the method of multipliers over dual ascent comes at a cost. When f is separable, the augmented Lagrangian L_ρ is not separable, so the x-minimization step (2.7) cannot be carried out separately in parallel for each x_i. This means that the basic method of multipliers cannot be used for decomposition. We will see how to address this issue next.

Augmented Lagrangians and the method of multipliers for constrained optimization were first proposed in the late 1960s by Hestenes [97, 98] and Powell [138]. Many of the early numerical experiments on the method of multipliers are due to Miele et al. [124, 125, 126]. Much of the early work is consolidated in a monograph by Bertsekas [15], who also discusses similarities to older approaches using Lagrangians and penalty functions [6, 5, 71], as well as a number of generalizations.

3

Alternating Direction Method of Multipliers

3.1 Algorithm

ADMM is an algorithm that is intended to blend the decomposability of dual ascent with the superior convergence properties of the method of multipliers. The algorithm solves problems in the form

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array} \tag{3.1}$$

with variables $x \in \mathbf{R}^n$ and $z \in \mathbf{R}^m$, where $A \in \mathbf{R}^{p\times n}$, $B \in \mathbf{R}^{p\times m}$, and $c \in \mathbf{R}^p$. We will assume that f and g are convex; more specific assumptions will be discussed in §3.2. The only difference from the general linear equality-constrained problem (2.1) is that the variable, called x there, has been split into two parts, called x and z here, with the objective function separable across this splitting. The optimal value of the problem (3.1) will be denoted by

$$p^\star = \inf\{f(x) + g(z) \mid Ax + Bz = c\}.$$

As in the method of multipliers, we form the augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2.$$

ADMM consists of the iterations

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \tag{3.2}$$

$$z^{k+1} := \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k) \tag{3.3}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \tag{3.4}$$

where $\rho > 0$. The algorithm is very similar to dual ascent and the method of multipliers: it consists of an x-minimization step (3.2), a z-minimization step (3.3), and a dual variable update (3.4). As in the method of multipliers, the dual variable update uses a step size equal to the augmented Lagrangian parameter ρ.

The method of multipliers for (3.1) has the form

$$\begin{aligned}(x^{k+1}, z^{k+1}) &:= \operatorname*{argmin}_{x,z} L_\rho(x, z, y^k)\\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c).\end{aligned}$$

Here the augmented Lagrangian is minimized jointly with respect to the two primal variables. In ADMM, on the other hand, x and z are updated in an alternating or sequential fashion, which accounts for the term *alternating direction*. ADMM can be viewed as a version of the method of multipliers where a single *Gauss-Seidel* pass [90, §10.1] over x and z is used instead of the usual joint minimization. Separating the minimization over x and z into two steps is precisely what allows for decomposition when f or g are separable.

The algorithm state in ADMM consists of z^k and y^k. In other words, (z^{k+1}, y^{k+1}) is a function of (z^k, y^k). The variable x^k is not part of the state; it is an intermediate result computed from the previous state (z^{k-1}, y^{k-1}).

If we switch (re-label) x and z, f and g, and A and B in the problem (3.1), we obtain a variation on ADMM with the order of the x-update step (3.2) and z-update step (3.3) reversed. The roles of x and z are almost symmetric, but not quite, since the dual update is done after the z-update but before the x-update.

3.1.1 Scaled Form

ADMM can be written in a slightly different form, which is often more convenient, by combining the linear and quadratic terms in the augmented Lagrangian and scaling the dual variable. Defining the residual $r = Ax + Bz - c$, we have

$$\begin{aligned} y^T r + (\rho/2)\|r\|_2^2 &= (\rho/2)\|r + (1/\rho)y\|_2^2 - (1/2\rho)\|y\|_2^2 \\ &= (\rho/2)\|r + u\|_2^2 - (\rho/2)\|u\|_2^2, \end{aligned}$$

where $u = (1/\rho)y$ is the *scaled dual variable*. Using the scaled dual variable, we can express ADMM as

$$x^{k+1} := \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|Ax + Bz^k - c + u^k\|_2^2 \right) \tag{3.5}$$

$$z^{k+1} := \operatorname*{argmin}_z \left(g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + u^k\|_2^2 \right) \tag{3.6}$$

$$u^{k+1} := u^k + Ax^{k+1} + Bz^{k+1} - c. \tag{3.7}$$

Defining the residual at iteration k as $r^k = Ax^k + Bz^k - c$, we see that

$$u^k = u^0 + \sum_{j=1}^{k} r^j,$$

the running sum of the residuals.

We call the first form of ADMM above, given by (3.2–3.4), the *unscaled form*, and the second form (3.5–3.7) the *scaled form*, since it is expressed in terms of a scaled version of the dual variable. The two are clearly equivalent, but the formulas in the scaled form of ADMM are often shorter than in the unscaled form, so we will use the scaled form in the sequel. We will use the unscaled form when we wish to emphasize the role of the dual variable or to give an interpretation that relies on the (unscaled) dual variable.

3.2 Convergence

There are many convergence results for ADMM discussed in the literature; here, we limit ourselves to a basic but still very general result that applies to all of the examples we will consider. We will make one

assumption about the functions f and g, and one assumption about problem (3.1).

Assumption 1. The (extended-real-valued) functions $f : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ and $g : \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$ are closed, proper, and convex.

This assumption can be expressed compactly using the epigraphs of the functions: The function f satisfies assumption 1 if and only if its epigraph

$$\mathbf{epi}\, f = \{(x,t) \in \mathbf{R}^n \times \mathbf{R} \mid f(x) \leq t\}$$

is a closed nonempty convex set.

Assumption 1 implies that the subproblems arising in the x-update (3.2) and z-update (3.3) are *solvable*, *i.e.*, there exist x and z, not necessarily unique (without further assumptions on A and B), that minimize the augmented Lagrangian. It is important to note that assumption 1 allows f and g to be nondifferentiable and to assume the value $+\infty$. For example, we can take f to be the indicator function of a closed nonempty convex set $\mathcal{C}$, *i.e.*, $f(x) = 0$ for $x \in \mathcal{C}$ and $f(x) = +\infty$ otherwise. In this case, the x-minimization step (3.2) will involve solving a constrained quadratic program over $\mathcal{C}$, the effective domain of f.

Assumption 2. The unaugmented Lagrangian L_0 has a saddle point.

Explicitly, there exist $(x^\star, z^\star, y^\star)$, not necessarily unique, for which

$$L_0(x^\star, z^\star, y) \leq L_0(x^\star, z^\star, y^\star) \leq L_0(x, z, y^\star)$$

holds for all $x,\ z,\ y$.

By assumption 1, it follows that $L_0(x^\star, z^\star, y^\star)$ is finite for any saddle point $(x^\star, z^\star, y^\star)$. This implies that $(x^\star, z^\star)$ is a solution to (3.1), so $Ax^\star + Bz^\star = c$ and $f(x^\star) < \infty$, $g(z^\star) < \infty$. It also implies that $y^\star$ is dual optimal, and the optimal values of the primal and dual problems are equal, *i.e.*, that strong duality holds. Note that we make no assumptions about A, B, or c, except implicitly through assumption 2; in particular, neither A nor B is required to be full rank.

3.2.1 Convergence

Under assumptions 1 and 2, the ADMM iterates satisfy the following:

- *Residual convergence.* $r^k \to 0$ as $k \to \infty$, *i.e.*, the iterates approach feasibility.
- *Objective convergence.* $f(x^k) + g(z^k) \to p^\star$ as $k \to \infty$, *i.e.*, the objective function of the iterates approaches the optimal value.
- *Dual variable convergence.* $y^k \to y^\star$ as $k \to \infty$, where $y^\star$ is a dual optimal point.

A proof of the residual and objective convergence results is given in appendix A. Note that x^k and z^k need not converge to optimal values, although such results can be shown under additional assumptions.

3.2.2 Convergence in Practice

Simple examples show that ADMM can be very slow to converge to high accuracy. However, it is often the case that ADMM converges to modest accuracy—sufficient for many applications—within a few tens of iterations. This behavior makes ADMM similar to algorithms like the conjugate gradient method, for example, in that a few tens of iterations will often produce acceptable results of practical use. However, the slow convergence of ADMM also distinguishes it from algorithms such as Newton's method (or, for constrained problems, interior-point methods), where high accuracy can be attained in a reasonable amount of time. While in some cases it is possible to combine ADMM with a method for producing a high accuracy solution from a low accuracy solution [64], in the general case ADMM will be practically useful mostly in cases when modest accuracy is sufficient. Fortunately, this is usually the case for the kinds of large-scale problems we consider. Also, in the case of statistical and machine learning problems, solving a parameter estimation problem to very high accuracy often yields little to no improvement in actual prediction performance, the real metric of interest in applications.

3.3 Optimality Conditions and Stopping Criterion

The necessary and sufficient optimality conditions for the ADMM problem (3.1) are primal feasibility,

$$Ax^\star + Bz^\star - c = 0, \tag{3.8}$$

and dual feasibility,

$$0 \in \partial f(x^\star) + A^T y^\star \tag{3.9}$$

$$0 \in \partial g(z^\star) + B^T y^\star. \tag{3.10}$$

Here, ∂ denotes the subdifferential operator; see, *e.g.*, [140, 19, 99]. (When f and g are differentiable, the subdifferentials ∂f and ∂g can be replaced by the gradients ∇f and ∇g, and $\in$ can be replaced by $=$.)

Since z^{k+1} minimizes $L_\rho(x^{k+1}, z, y^k)$ by definition, we have that

$$\begin{aligned} 0 &\in \partial g(z^{k+1}) + B^T y^k + \rho B^T (Ax^{k+1} + Bz^{k+1} - c) \\ &= \partial g(z^{k+1}) + B^T y^k + \rho B^T r^{k+1} \\ &= \partial g(z^{k+1}) + B^T y^{k+1}. \end{aligned}$$

This means that z^{k+1} and y^{k+1} always satisfy (3.10), so attaining optimality comes down to satisfying (3.8) and (3.9). This phenomenon is analogous to the iterates of the method of multipliers always being dual feasible; see page 11.

Since x^{k+1} minimizes $L_\rho(x, z^k, y^k)$ by definition, we have that

$$\begin{aligned} 0 &\in \partial f(x^{k+1}) + A^T y^k + \rho A^T (Ax^{k+1} + Bz^k - c) \\ &= \partial f(x^{k+1}) + A^T (y^k + \rho r^{k+1} + \rho B(z^k - z^{k+1})) \\ &= \partial f(x^{k+1}) + A^T y^{k+1} + \rho A^T B(z^k - z^{k+1}), \end{aligned}$$

or equivalently,

$$\rho A^T B(z^{k+1} - z^k) \in \partial f(x^{k+1}) + A^T y^{k+1}.$$

This means that the quantity

$$s^{k+1} = \rho A^T B(z^{k+1} - z^k)$$

can be viewed as a residual for the dual feasibility condition (3.9). We will refer to s^{k+1} as the *dual residual* at iteration $k+1$, and to $r^{k+1} = Ax^{k+1} + Bz^{k+1} - c$ as the *primal residual* at iteration $k+1$.

In summary, the optimality conditions for the ADMM problem consist of three conditions, (3.8–3.10). The last condition (3.10) always holds for $(x^{k+1}, z^{k+1}, y^{k+1})$; the residuals for the other two, (3.8) and (3.9), are the primal and dual residuals r^{k+1} and s^{k+1}, respectively. These two residuals converge to zero as ADMM proceeds. (In fact, the convergence proof in appendix A shows $B(z^{k+1} - z^k)$ converges to zero, which implies s^k converges to zero.)

3.3.1 Stopping Criteria

The residuals of the optimality conditions can be related to a bound on the objective suboptimality of the current point, *i.e.*, $f(x^k) + g(z^k) - p^\star$. As shown in the convergence proof in appendix A, we have

$$f(x^k) + g(z^k) - p^\star \leq -(y^k)^T r^k + (x^k - x^\star)^T s^k. \tag{3.11}$$

This shows that when the residuals r^k and s^k are small, the objective suboptimality also must be small. We cannot use this inequality directly in a stopping criterion, however, since we do not know $x^\star$. But if we guess or estimate that $\|x^k - x^\star\|_2 \leq d$, we have that

$$f(x^k) + g(z^k) - p^\star \leq -(y^k)^T r^k + d\|s^k\|_2 \leq \|y^k\|_2\|r^k\|_2 + d\|s^k\|_2.$$

The middle or righthand terms can be used as an approximate bound on the objective suboptimality (which depends on our guess of d).

This suggests that a reasonable termination criterion is that the primal and dual residuals must be small, *i.e.*,

$$\|r^k\|_2 \leq \epsilon^{\text{pri}} \quad \text{and} \quad \|s^k\|_2 \leq \epsilon^{\text{dual}}, \tag{3.12}$$

where $\epsilon^{\text{pri}} > 0$ and $\epsilon^{\text{dual}} > 0$ are feasibility tolerances for the primal and dual feasibility conditions (3.8) and (3.9), respectively. These tolerances can be chosen using an absolute and relative criterion, such as

$$\begin{aligned} \epsilon^{\text{pri}} &= \sqrt{p}\,\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|Ax^k\|_2, \|Bz^k\|_2, \|c\|_2\}, \\ \epsilon^{\text{dual}} &= \sqrt{n}\,\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|A^T y^k\|_2, \end{aligned}$$

where $\epsilon^{\text{abs}} > 0$ is an absolute tolerance and $\epsilon^{\text{rel}} > 0$ is a relative tolerance. (The factors $\sqrt{p}$ and $\sqrt{n}$ account for the fact that the ℓ_2 norms are in $\mathbf{R}^p$ and $\mathbf{R}^n$, respectively.) A reasonable value for the relative stopping

criterion might be $\epsilon^{\text{rel}} = 10^{-3}$ or 10^{-4}, depending on the application. The choice of absolute stopping criterion depends on the scale of the typical variable values.

3.4 Extensions and Variations

Many variations on the classic ADMM algorithm have been explored in the literature. Here we briefly survey some of these variants, organized into groups of related ideas. Some of these methods can give superior convergence in practice compared to the standard ADMM presented above. Most of the extensions have been rigorously analyzed, so the convergence results described above are still valid (in some cases, under some additional conditions).

3.4.1 Varying Penalty Parameter

A standard extension is to use possibly different penalty parameters ρ^k for each iteration, with the goal of improving the convergence in practice, as well as making performance less dependent on the initial choice of the penalty parameter. In the context of the method of multipliers, this approach is analyzed in [142], where it is shown that superlinear convergence may be achieved if $\rho^k \to \infty$. Though it can be difficult to prove the convergence of ADMM when ρ varies by iteration, the fixed-ρ theory still applies if one just assumes that ρ becomes fixed after a finite number of iterations.

A simple scheme that often works well is (see, *e.g.*, [96, 169]):

$$\rho^{k+1} := \begin{cases} \tau^{\text{incr}}\rho^k & \text{if } \|r^k\|_2 > \mu\|s^k\|_2 \\ \rho^k/\tau^{\text{decr}} & \text{if } \|s^k\|_2 > \mu\|r^k\|_2 \\ \rho^k & \text{otherwise,} \end{cases} \tag{3.13}$$

where $\mu > 1$, $\tau^{\text{incr}} > 1$, and $\tau^{\text{decr}} > 1$ are parameters. Typical choices might be $\mu = 10$ and $\tau^{\text{incr}} = \tau^{\text{decr}} = 2$. The idea behind this penalty parameter update is to try to keep the primal and dual residual norms within a factor of μ of one another as they both converge to zero.

The ADMM update equations suggest that large values of ρ place a large penalty on violations of primal feasibility and so tend to produce

small primal residuals. Conversely, the definition of s^{k+1} suggests that small values of ρ tend to reduce the dual residual, but at the expense of reducing the penalty on primal feasibility, which may result in a larger primal residual. The adjustment scheme (3.13) inflates ρ by τ^{incr} when the primal residual appears large compared to the dual residual, and deflates ρ by τ^{decr} when the primal residual seems too small relative to the dual residual. This scheme may also be refined by taking into account the relative magnitudes of ϵ^{pri} and ϵ^{dual}.

When a varying penalty parameter is used in the scaled form of ADMM, the scaled dual variable $u^k = (1/\rho)y^k$ must also be rescaled after updating ρ; for example, if ρ is halved, u^k should be doubled before proceeding.

3.4.2 More General Augmenting Terms

Another idea is to allow for a different penalty parameter for each constraint, or more generally, to replace the quadratic term $(\rho/2)\|r\|_2^2$ with $(1/2)r^T Pr$, where P is a symmetric positive definite matrix. When P is constant, we can interpret this generalized version of ADMM as standard ADMM applied to a modified initial problem with the equality constraints $r = 0$ replaced with $Fr = 0$, where $F^T F = P$.

3.4.3 Over-relaxation

In the z- and y-updates, the quantity Ax^{k+1} can be replaced with

$$\alpha^k Ax^{k+1} - (1 - \alpha^k)(Bz^k - c),$$

where $\alpha^k \in (0, 2)$ is a *relaxation parameter*; when $\alpha^k > 1$, this technique is called *over-relaxation*, and when $\alpha^k < 1$, it is called *under-relaxation*. This scheme is analyzed in [63], and experiments in [59, 64] suggest that over-relaxation with $\alpha^k \in [1.5, 1.8]$ can improve convergence.

3.4.4 Inexact Minimization

ADMM will converge even when the x- and z-minimization steps are not carried out exactly, provided certain suboptimality measures

in the minimizations satisfy an appropriate condition, such as being summable. This result is due to Eckstein and Bertsekas [63], building on earlier results by Gol'shtein and Tret'yakov [89]. This technique is important in situations where the x- or z-updates are carried out using an iterative method; it allows us to solve the minimizations only approximately at first, and then more accurately as the iterations progress.

3.4.5 Update Ordering

Several variations on ADMM involve performing the x-, z-, and y-updates in varying orders or multiple times. For example, we can divide the variables into k blocks, and update each of them in turn, possibly multiple times, before performing each dual variable update; see, *e.g.*, [146]. Carrying out multiple x- and z-updates before the y-update can be interpreted as executing multiple Gauss-Seidel passes instead of just one; if many sweeps are carried out before each dual update, the resulting algorithm is very close to the standard method of multipliers [17, §3.4.4]. Another variation is to perform an additional y-update between the x- and z-update, with half the step length [17].

3.4.6 Related Algorithms

There are also a number of other algorithms distinct from but inspired by ADMM. For instance, Fukushima [80] applies ADMM to a dual problem formulation, yielding a 'dual ADMM' algorithm, which is shown in [65] to be equivalent to the 'primal Douglas-Rachford' method discussed in [57, §3.5.6]. As another example, Zhu et al. [183] discuss variations of distributed ADMM (discussed in §7, §8, and §10) that can cope with various complicating factors, such as noise in the messages exchanged for the updates, or asynchronous updates, which can be useful in a regime when some processors or subsystems randomly fail. There are also algorithms resembling a combination of ADMM and the *proximal* method of multipliers [141], rather than the standard method of multipliers; see, *e.g.*, [33, 60]. Other representative publications include [62, 143, 59, 147, 158, 159, 42].

3.5 Notes and References

ADMM was originally proposed in the mid-1970s by Glowinski and Marrocco [86] and Gabay and Mercier [82]. There are a number of other important papers analyzing the properties of the algorithm, including [76, 81, 75, 87, 157, 80, 65, 33]. In particular, the convergence of ADMM has been explored by many authors, including Gabay [81] and Eckstein and Bertsekas [63].

ADMM has also been applied to a number of statistical problems, such as constrained sparse regression [18], sparse signal recovery [70], image restoration and denoising [72, 154, 134], trace norm regularized least squares minimization [174], sparse inverse covariance selection [178], the Dantzig selector [116], and support vector machines [74], among others. For examples in signal processing, see [42, 40, 41, 150, 149] and the references therein.

Many papers analyzing ADMM do so from the perspective of *maximal monotone operators* [23, 141, 142, 63, 144]. Briefly, a wide variety of problems can be posed as finding a zero of a maximal monotone operator; for example, if f is closed, proper, and convex, then the subdifferential operator ∂f is maximal monotone, and finding a zero of ∂f is simply minimization of f; such a minimization may implicitly contain constraints if f is allowed to take the value $+\infty$. Rockafellar's *proximal point algorithm* [142] is a general method for finding a zero of a maximal monotone operator, and a wide variety of algorithms have been shown to be special cases, including proximal minimization (see §4.1), the method of multipliers, and ADMM. For a more detailed review of the older literature, see [57, §2].

The method of multipliers was shown to be a special case of the proximal point algorithm by Rockafellar [141]. Gabay [81] showed that ADMM is a special case of a method called *Douglas-Rachford splitting* for monotone operators [53, 112], and Eckstein and Bertsekas [63] showed in turn that Douglas-Rachford splitting is a special case of the proximal point algorithm. (The variant of ADMM that performs an extra y-update between the x- and z-updates is equivalent to *Peaceman-Rachford splitting* [137, 112] instead, as shown by Glowinski and Le Tallec [87].) Using the same framework, Eckstein

and Bertsekas [63] also showed the relationships between a number of other algorithms, such as Spingarn's method of partial inverses [153]. Lawrence and Spingarn [108] develop an alternative framework showing that Douglas-Rachford splitting, hence ADMM, is a special case of the proximal point algorithm; Eckstein and Ferris [64] offer a more recent discussion explaining this approach.

The major importance of these results is that they allow the powerful convergence theory for the proximal point algorithm to apply directly to ADMM and other methods, and show that many of these algorithms are essentially identical. (But note that our proof of convergence of the basic ADMM algorithm, given in appendix A, is self-contained and does not rely on this abstract machinery.) Research on operator splitting methods and their relation to decomposition algorithms continues to this day [66, 67].

A considerable body of recent research considers replacing the quadratic penalty term in the standard method of multipliers with a more general deviation penalty, such as one derived from a *Bregman divergence* [30, 58]; see [22] for background material. Unfortunately, these generalizations do not appear to carry over in a straightforward manner from non-decomposition augmented Lagrangian methods to ADMM: There is currently no proof of convergence known for ADMM with nonquadratic penalty terms.

4

General Patterns

Structure in f, g, A, and B can often be exploited to carry out the x- and z-updates more efficiently. Here we consider three general cases that we will encounter repeatedly in the sequel: quadratic objective terms, separable objective and constraints, and smooth objective terms. Our discussion will be written for the x-update but applies to the z-update by symmetry. We express the x-update step as

$$x^+ = \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|Ax - v\|_2^2\right),$$

where $v = -Bz + c - u$ is a known constant vector for the purposes of the x-update.

4.1 Proximity Operator

First, consider the simple case where $A = I$, which appears frequently in the examples. Then the x-update is

$$x^+ = \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - v\|_2^2\right).$$

As a function of v, the righthand side is denoted $\mathbf{prox}_{f,\rho}(v)$ and is called the *proximity operator* of f with penalty ρ [127]. In variational

analysis,

$$\tilde{f}(v) = \inf_x \left(f(x) + (\rho/2)\|x - v\|_2^2\right)$$

is known as the *Moreau envelope* or *Moreau-Yosida regularization* of f, and is connected to the theory of the proximal point algorithm [144]. The x-minimization in the proximity operator is generally referred to as *proximal minimization*. While these observations do not by themselves allow us to improve the efficiency of ADMM, it does tie the x-minimization step to other well known ideas.

When the function f is simple enough, the x-update (*i.e.*, the proximity operator) can be evaluated analytically; see [41] for many examples. For instance, if f is the indicator function of a closed nonempty convex set $\mathcal{C}$, then the x-update is

$$x^+ = \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - v\|_2^2\right) = \Pi_{\mathcal{C}}(v),$$

where $\Pi_{\mathcal{C}}$ denotes projection (in the Euclidean norm) onto $\mathcal{C}$. This holds independently of the choice of ρ. As an example, if f is the indicator function of the nonnegative orthant $\mathbf{R}^n_+$, we have $x^+ = (v)_+$, the vector obtained by taking the nonnegative part of each component of v.

4.2 Quadratic Objective Terms

Suppose f is given by the (convex) quadratic function

$$f(x) = (1/2)x^T P x + q^T x + r,$$

where $P \in \mathbf{S}^n_+$, the set of symmetric positive semidefinite $n \times n$ matrices. This includes the cases when f is linear or constant, by setting P, or both P and q, to zero. Then, assuming $P + \rho A^T A$ is invertible, x^+ is an affine function of v given by

$$x^+ = (P + \rho A^T A)^{-1}(\rho A^T v - q). \tag{4.1}$$

In other words, computing the x-update amounts to solving a linear system with positive definite coefficient matrix $P + \rho A^T A$ and right-hand side $\rho A^T v - q$. As we show below, an appropriate use of numerical linear algebra can exploit this fact and substantially improve performance. For general background on numerical linear algebra, see [47] or [90]; see [20, appendix C] for a short overview of direct methods.

4.2.1 Direct Methods

We assume here that a *direct method* is used to carry out the x-update; the case when an iterative method is used is discussed in §4.3. Direct methods for solving a linear system $Fx = g$ are based on first *factoring* $F = F_1F_2 \cdots F_k$ into a product of simpler matrices, and then computing $x = F^{-1}b$ by *solving* a sequence of problems of the form $F_iz_i = z_{i-1}$, where $z_1 = F_1^{-1}g$ and $x = z_k$. The solve step is sometimes also called a *back-solve*. The computational cost of factorization and back-solve operations depends on the sparsity pattern and other properties of F. The cost of solving $Fx = g$ is the sum of the cost of factoring F and the cost of the back-solve.

In our case, the coefficient matrix is $F = P + \rho A^TA$ and the right-hand side is $g = \rho A^Tv - q$, where $P \in \mathbf{S}_+^n$ and $A \in \mathbf{R}^{p\times n}$. Suppose we exploit no structure in A or P, *i.e.*, we use generic methods that work for any matrix. We can form $F = P + \rho A^TA$ at a cost of $O(pn^2)$ flops (floating point operations). We then carry out a Cholesky factorization of F at a cost of $O(n^3)$ flops; the back-solve cost is $O(n^2)$. (The cost of forming g is negligible compared to the costs listed above.) When p is on the order of, or more than n, the overall cost is $O(pn^2)$. (When p is less than n in order, the matrix inversion lemma described below can be used to carry out the update in $O(p^2n)$ flops.)

4.2.2 Exploiting Sparsity

When A and P are such that F is sparse (*i.e.*, has enough zero entries to be worth exploiting), much more efficient factorization and back-solve routines can be employed. As an extreme case, if P and A are diagonal $n \times n$ matrices, then both the factor and solve costs are $O(n)$. If P and A are banded, then so is F. If F is banded with bandwidth k, the factorization cost is $O(nk^2)$ and the back-solve cost is $O(nk)$. In this case, the x-update can be carried out at a cost $O(nk^2)$, plus the cost of forming F. The same approach works when $P + \rho A^TA$ has a more general sparsity pattern; in this case, a permuted Cholesky factorization can be used, with permutations chosen to reduce fill-in.

4.2.3 Caching Factorizations

Now suppose we need to solve multiple linear systems, say, $Fx^{(i)} = g^{(i)}$, $i = 1, \ldots, N$, with the same coefficient matrix but different righthand sides. This occurs in ADMM when the parameter ρ is not changed. In this case, the factorization of the coefficient matrix F can be computed once and then back-solves can be carried out for each righthand side. If t is the factorization cost and s is the back-solve cost, then the total cost becomes $t + Ns$ instead of $N(t + s)$, which would be the cost if we were to factor F each iteration. As long as ρ does not change, we can factor $P + \rho A^T A$ *once*, and then use this cached factorization in subsequent solve steps. For example, if we do not exploit any structure and use the standard Cholesky factorization, the x-update steps can be carried out a factor n more efficiently than a naive implementation, in which we solve the equations from scratch in each iteration.

When structure is exploited, the ratio between t and s is typically less than n but often still significant, so here too there are performance gains. However, in this case, there is less benefit to ρ not changing, so we can weigh the benefit of varying ρ against the benefit of not recomputing the factorization of $P + \rho A^T A$. In general, an implementation should cache the factorization of $P + \rho A^T A$ and then only recompute it if and when ρ changes.

4.2.4 Matrix Inversion Lemma

We can also exploit structure using the *matrix inversion lemma*, which states that the identity

$$(P + \rho A^T A)^{-1} = P^{-1} - \rho P^{-1} A^T (I + \rho A P^{-1} A^T)^{-1} A P^{-1}$$

holds when all the inverses exist. This means that if linear systems with coefficient matrix P can be solved efficiently, and p is small, or at least no larger than n in order, then the x-update can be computed efficiently as well. The same trick as above can also be used to obtain an efficient method for computing multiple updates: The factorization of $I + \rho A P^{-1} A^T$ can be cached and cheaper back-solves can be used in computing the updates.

As an example, suppose that P is diagonal and that $p \leq n$. A naive method for computing the update costs $O(n^3)$ flops in the first iteration and $O(n^2)$ flops in subsequent iterations, if we store the factors of $P + \rho A^T A$. Using the matrix inversion lemma (*i.e.*, using the righthand side above) to compute the x-update costs $O(np^2)$ flops, an improvement by a factor of $(n/p)^2$ over the naive method. In this case, the dominant cost is forming $AP^{-1}A^T$. The factors of $I + \rho AP^{-1}A^T$ can be saved after the first update, so subsequent iterations can be carried out at cost $O(np)$ flops, a savings of a factor of p over the first update.

Using the matrix inversion lemma to compute x^+ can also make it less costly to vary ρ in each iteration. When P is diagonal, for example, we can compute $AP^{-1}A^T$ once, and then form and factor $I + \rho^k AP^{-1}A^T$ in iteration k at a cost of $O(p^3)$ flops. In other words, the update costs an additional $O(np)$ flops, so if p^2 is less than or equal to n in order, there is no additional cost (in order) to carrying out updates with ρ varying in each iteration.

4.2.5 Quadratic Function Restricted to an Affine Set

The same comments hold for the slightly more complex case of a convex quadratic function restricted to an affine set:

$$f(x) = (1/2)x^T Px + q^T x + r, \qquad \mathbf{dom}\, f = \{x \mid Fx = g\}.$$

Here, x^+ is still an affine function of v, and the update involves solving the KKT (Karush-Kuhn-Tucker) system

$$\begin{bmatrix} P + \rho I & F^T \\ F & 0 \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu \end{bmatrix} + \begin{bmatrix} q - \rho(z^k - u^k) \\ -g \end{bmatrix} = 0.$$

All of the comments above hold here as well: Factorizations can be cached to carry out additional updates more efficiently, and structure in the matrices can be exploited to improve the efficiency of the factorization and back-solve steps.

4.3 Smooth Objective Terms

4.3.1 Iterative Solvers

When f is smooth, general iterative methods can be used to carry out the x-minimization step. Of particular interest are methods that only require the ability to compute $\nabla f(x)$ for a given x, to multiply a vector by A, and to multiply a vector by A^T. Such methods can scale to relatively large problems. Examples include the standard gradient method, the (nonlinear) conjugate gradient method, and the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [113, 26]; see [135] for further details.

The convergence of these methods depends on the conditioning of the function to be minimized. The presence of the quadratic penalty term $(\rho/2)\|Ax - v\|_2^2$ tends to improve the conditioning of the problem and thus improve the performance of an iterative method for updating x. Indeed, one method for adjusting the parameter ρ from iteration to iteration is to increase it until the iterative method used to carry out the updates converges quickly enough.

4.3.2 Early Termination

A standard technique to speed up the algorithm is to terminate the iterative method used to carry out the x-update (or z-update) early, *i.e.*, before the gradient of $f(x) + (\rho/2)\|Ax - v\|_2^2$ is very small. This technique is justified by the convergence results for ADMM with inexact minimization in the x- and z-update steps. In this case, the required accuracy should be low in the initial iterations of ADMM and then repeatedly increased in each iteration. Early termination in the x- or z-updates can result in more ADMM iterations, but much lower cost per iteration, giving an overall improvement in efficiency.

4.3.3 Warm Start

Another standard trick is to initialize the iterative method used in the x-update at the solution x^k obtained in the previous iteration. This is called a *warm start*. The previous ADMM iterate often gives a good enough approximation to result in far fewer iterations (of the

iterative method used to compute the update x^{k+1}) than if the iterative method were started at zero or some other default initialization. This is especially the case when ADMM has almost converged, in which case the updates will not change significantly from their previous values.

4.3.4 Quadratic Objective Terms

Even when f is quadratic, it may be worth using an iterative method rather than a direct method for the x-update. In this case, we can use a standard (possibly preconditioned) conjugate gradient method. This approach makes sense when direct methods do not work (*e.g.*, because they require too much memory) or when A is dense but a fast method is available for multiplying a vector by A or A^T. This is the case, for example, when A represents the discrete Fourier transform [90].

4.4 Decomposition

4.4.1 Block Separability

Suppose $x = (x_1, \ldots, x_N)$ is a partition of the variable x into subvectors and that f is separable with respect to this partition, *i.e.*,

$$f(x) = f_1(x_1) + \cdots + f_N(x_N),$$

where $x_i \in \mathbf{R}^{n_i}$ and $\sum_{i=1}^N n_i = N$. If the quadratic term $\|Ax\|_2^2$ is also separable with respect to the partition, *i.e.*, A^TA is block diagonal conformably with the partition, then the augmented Lagrangian L_ρ is separable. This means that the x-update can be carried out in parallel, with the subvectors x_i updated by N separate minimizations.

4.4.2 Component Separability

In some cases, the decomposition extends all the way to individual components of x, *i.e.*,

$$f(x) = f_1(x_1) + \cdots + f_n(x_n),$$

where $f_i : \mathbf{R} \to \mathbf{R}$, and A^TA is diagonal. The x-minimization step can then be carried out via n *scalar* minimizations, which can in some cases be expressed analytically (but in any case can be computed very efficiently). We will call this *component separability*.

4.4.3 Soft Thresholding

For an example that will come up often in the sequel, consider $f(x) = \lambda\|x\|_1$ (with $\lambda > 0$) and $A = I$. In this case the (scalar) x_i-update is

$$x_i^+ := \operatorname*{argmin}_{x_i} \left(\lambda|x_i| + (\rho/2)(x_i - v_i)^2\right).$$

Even though the first term is not differentiable, we can easily compute a simple closed-form solution to this problem by using subdifferential calculus; see [140, §23] for background. Explicitly, the solution is

$$x_i^+ := S_{\lambda/\rho}(v_i),$$

where the *soft thresholding operator* S is defined as

$$S_\kappa(a) = \begin{cases} a - \kappa & a > \kappa \\ 0 & |a| \leq \kappa \\ a + \kappa & a < -\kappa, \end{cases}$$

or equivalently,

$$S_\kappa(a) = (a - \kappa)_+ - (-a - \kappa)_+.$$

Yet another formula, which shows that the soft thresholding operator is a shrinkage operator (*i.e.*, moves a point toward zero), is

$$S_\kappa(a) = (1 - \kappa/|a|)_+ a \tag{4.2}$$

(for $a \neq 0$). We refer to updates that reduce to this form as *element-wise soft thresholding*. In the language of §4.1, soft thresholding is the proximity operator of the ℓ_1 norm.

5

Constrained Convex Optimization

The generic constrained convex optimization problem is

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{C}, \end{array} \tag{5.1}$$

with variable $x \in \mathbf{R}^n$, where f and $\mathcal{C}$ are convex. This problem can be rewritten in ADMM form (3.1) as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array}$$

where g is the indicator function of $\mathcal{C}$.

The augmented Lagrangian (using the scaled dual variable) is

$$L_\rho(x, z, u) = f(x) + g(z) + (\rho/2)\|x - z + u\|_2^2,$$

so the scaled form of ADMM for this problem is

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - z^k + u^k\|_2^2 \right) \\ z^{k+1} &:= \Pi_{\mathcal{C}}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

The x-update involves minimizing f plus a convex quadratic function, *i.e.*, evaluation of the proximal operator associated with f. The z-update is Euclidean projection onto $\mathcal{C}$. The objective f need not be smooth here; indeed, we can include additional constraints (*i.e.*, beyond those represented by $x \in \mathcal{C}$) by defining f to be $+\infty$ where they are violated. In this case, the x-update becomes a constrained optimization problem over $\mathbf{dom}\, f = \{x \mid f(x) < \infty\}$.

As with all problems where the constraint is $x - z = 0$, the primal and dual residuals take the simple form

$$r^k = x^k - z^k, \qquad s^k = -\rho(z^k - z^{k-1}).$$

In the following sections we give some more specific examples.

5.1 Convex Feasibility

5.1.1 Alternating Projections

A classic problem is to find a point in the intersection of two closed nonempty convex sets. The classical method, which dates back to the 1930s, is von Neumann's *alternating projections* algorithm [166, 36, 21]:

$$\begin{aligned} x^{k+1} &:= \Pi_{\mathcal{C}}(z^k) \\ z^{k+1} &:= \Pi_{\mathcal{D}}(x^{k+1}), \end{aligned}$$

where $\Pi_{\mathcal{C}}$ and $\Pi_{\mathcal{D}}$ are Euclidean projection onto the sets $\mathcal{C}$ and $\mathcal{D}$, respectively.

In ADMM form, the problem can be written as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array}$$

where f is the indicator function of $\mathcal{C}$ and g is the indicator function of $\mathcal{D}$. The scaled form of ADMM is then

$$\begin{aligned} x^{k+1} &:= \Pi_{\mathcal{C}}(z^k - u^k) \\ z^{k+1} &:= \Pi_{\mathcal{D}}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}, \end{aligned} \tag{5.2}$$

so both the x and z steps involve projection onto a convex set, as in the classical method. This is exactly Dykstra's alternating projections

method [56, 9], which is far more efficient than the classical method that does not use the dual variable u.

Here, the norm of the primal residual $\|x^k - z^k\|_2$ has a nice interpretation. Since $x^k \in \mathcal{C}$ and $z^k \in \mathcal{D}$, $\|x^k - z^k\|_2$ is an upper bound on $\mathbf{dist}(\mathcal{C}, \mathcal{D})$, the Euclidean distance between $\mathcal{C}$ and $\mathcal{D}$. If we terminate with $\|r^k\|_2 \leq \epsilon^{\text{pri}}$, then we have found a pair of points, one in $\mathcal{C}$ and one in $\mathcal{D}$, that are no more than ϵ^{pri} far apart. Alternatively, the point $(1/2)(x^k + z^k)$ is no more than a distance $\epsilon^{\text{pri}}/2$ from both $\mathcal{C}$ and $\mathcal{D}$.

5.1.2 Parallel Projections

This method can be applied to the problem of finding a point in the intersection of N closed convex sets $\mathcal{A}_1, \ldots, \mathcal{A}_N$ in $\mathbf{R}^n$ by running the algorithm in $\mathbf{R}^{nN}$ with

$$\mathcal{C} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N, \quad \mathcal{D} = \{(x_1, \ldots, x_N) \in \mathbf{R}^{nN} \mid x_1 = x_2 = \cdots = x_N\}.$$

If $x = (x_1, \ldots, x_N) \in \mathbf{R}^{nN}$, then projection onto $\mathcal{C}$ is

$$\Pi_{\mathcal{C}}(x) = (\Pi_{\mathcal{A}_1}(x_1), \ldots, \Pi_{\mathcal{A}_N}(x_N)),$$

and projection onto $\mathcal{D}$ is

$$\Pi_{\mathcal{D}}(x) = (\overline{x}, \overline{x}, \ldots, \overline{x}),$$

where $\overline{x} = (1/N)\sum_{i=1}^N x_i$ is the average of $x_1, \ldots, x_N$. Thus, each step of ADMM can be carried out by projecting points onto each of the sets $\mathcal{A}_i$ in parallel and then averaging the results:

$$\begin{aligned} x_i^{k+1} &:= \Pi_{\mathcal{A}_i}(z^k - u_i^k) \\ z^{k+1} &:= \frac{1}{N}\sum_{i=1}^N (x_i^{k+1} + u_i^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

Here the first and third steps are carried out in parallel, for $i = 1, \ldots, N$. (The description above involves a small abuse of notation in dropping the index i from z_i, since they are all the same.) This can be viewed as a special case of constrained optimization, as described in §4.4, where the indicator function of $\mathcal{A}_1 \cap \cdots \cap \mathcal{A}_N$ splits into the sum of the indicator functions of each $\mathcal{A}_i$.

We note for later reference a simplification of the ADMM algorithm above. Taking the average (over i) of the last equation we obtain

$$\overline{u}^{k+1} = \overline{u}^k + \overline{x}^{k+1} - z^k,$$

combined with $z^{k+1} = \overline{x}^{k+1} + \overline{u}^k$ (from the second equation) we see that $\overline{u}^{k+1} = 0$. So after the first step, the average of u_i is zero. This means that z^{k+1} reduces to $\overline{x}^{k+1}$. Using these simplifications, we arrive at the simple algorithm

$$\begin{aligned} x_i^{k+1} &:= \Pi_{\mathcal{A}_i}(\overline{x}^k - u_i^k) \\ u_i^{k+1} &:= u_i^k + (x_i^{k+1} - \overline{x}^{k+1}). \end{aligned}$$

This shows that u_i^k is the running sum of the the 'discrepancies' $x_i^k - \overline{x}^k$ (assuming $u^0 = 0$). The first step is a parallel projection onto the sets $\mathcal{C}_i$; the second involves averaging the projected points.

There is a large literature on successive projection algorithms and their many applications; see the survey by Bauschke and Borwein [10] for a general overview, Combettes [39] for applications to image processing, and Censor and Zenios [31, §5] for a discussion in the context of parallel optimization.

5.2 Linear and Quadratic Programming

The standard form quadratic program (QP) is

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = b, \quad x \geq 0, \end{array} \tag{5.3}$$

with variable $x \in \mathbf{R}^n$; we assume that $P \in \mathbf{S}_+^n$. When $P = 0$, this reduces to the standard form linear program (LP).

We express it in ADMM form as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array}$$

where

$$f(x) = (1/2)x^T P x + q^T x, \qquad \mathbf{dom}\, f = \{x \mid Ax = b\}$$

is the original objective with restricted domain and g is the indicator function of the nonnegative orthant $\mathbf{R}_+^n$.

The scaled form of ADMM consists of the iterations

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - z^k + u^k\|_2^2 \right) \\ z^{k+1} &:= (x^{k+1} + u^k)_+ \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

As described in §4.2.5, the x-update is an equality-constrained least squares problem with optimality conditions

$$\begin{bmatrix} P + \rho I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu \end{bmatrix} + \begin{bmatrix} q - \rho(z^k - u^k) \\ -b \end{bmatrix} = 0.$$

All of the comments on efficient computation from §4.2, such as storing factorizations so that subsequent iterations can be carried out cheaply, also apply here. For example, if P is diagonal, possibly zero, the first x-update can be carried out at a cost of $O(np^2)$ flops, where p is the number of equality constraints in the original quadratic program. Subsequent updates only cost $O(np)$ flops.

5.2.1 Linear and Quadratic Cone Programming

More generally, any conic constraint $x \in \mathcal{K}$ can be used in place of the constraint $x \geq 0$, in which case the standard quadratic program (5.3) becomes a quadratic conic program. The only change to ADMM is in the z-update, which then involves projection onto $\mathcal{K}$. For example, we can solve a semidefinite program with $x \in \mathbf{S}_+^n$ by projecting $x^{k+1} + u^k$ onto $\mathbf{S}_+^n$ using an eigenvalue decomposition.

6

ℓ_1-Norm Problems

The problems addressed in this section will help illustrate why ADMM is a natural fit for machine learning and statistical problems in particular. The reason is that, unlike dual ascent or the method of multipliers, ADMM explicitly targets problems that split into two distinct parts, f and g, that can then be handled separately. Problems of this form are pervasive in machine learning, because a significant number of learning problems involve minimizing a loss function together with a regularization term or side constraints. In other cases, these side constraints are introduced through problem transformations like putting the problem in consensus form, as will be discussed in §7.1.

This section contains a variety of simple but important problems involving ℓ_1 norms. There is widespread current interest in many of these problems across statistics, machine learning, and signal processing, and applying ADMM yields interesting algorithms that are state-of-the-art, or closely related to state-of-the-art methods. We will see that ADMM naturally lets us decouple the nonsmooth ℓ_1 term from the smooth loss term, which is computationally advantageous. In this section, we focus on the non-distributed versions of these problems for simplicity; the problem of distributed model fitting will be treated in the sequel.

The idea of ℓ_1 regularization is decades old, and traces back to Huber's [100] work on robust statistics and a paper of Claerbout [38] in geophysics. There is a vast literature, but some important modern papers are those on total variation denoising [145], soft thresholding [49], the lasso [156], basis pursuit [34], compressed sensing [50, 28, 29], and structure learning of sparse graphical models [123].

Because of the now widespread use of models incorporating an ℓ_1 penalty, there has also been considerable research on optimization algorithms for such problems. A recent survey by Yang et al. [173] compares and benchmarks a number of representative algorithms, including gradient projection [73, 102], homotopy methods [52], iterative shrinkage-thresholding [45], proximal gradient [132, 133, 11, 12], augmented Lagrangian methods [175], and interior-point methods [103]. There are other approaches as well, such as Bregman iterative algorithms [176] and iterative thresholding algorithms [51] implementable in a message-passing framework.

6.1 Least Absolute Deviations

A simple variant on least squares fitting is *least absolute deviations*, in which we minimize $\|Ax - b\|_1$ instead of $\|Ax - b\|_2^2$. Least absolute deviations provides a more robust fit than least squares when the data contains large outliers, and has been used extensively in statistics and econometrics. See, for example, [95, §10.6], [171, §9.6], and [20, §6.1.2].

In ADMM form, the problem can be written as

$$\begin{array}{ll} \text{minimize} & \|z\|_1 \\ \text{subject to} & Ax - z = b, \end{array}$$

so $f = 0$ and $g = \|\cdot\|_1$. Exploiting the special form of f and g, and assuming $A^T A$ is invertible, ADMM can be expressed as

$$\begin{aligned} x^{k+1} &:= (A^T A)^{-1} A^T (b + z^k - u^k) \\ z^{k+1} &:= S_{1/\rho}(Ax^{k+1} - b + u^k) \\ u^{k+1} &:= u^k + Ax^{k+1} - z^{k+1} - b, \end{aligned}$$

where the soft thresholding operator is interpreted elementwise. As in §4.2, the matrix $A^T A$ can be factored once; the factors are then used in cheaper back-solves in subsequent x-updates.

The x-update step is the same as carrying out a least squares fit with coefficient matrix A and righthand side $b + z^k - u^k$. Thus ADMM can be interpreted as a method for solving a least absolute deviations problem by iteratively solving the associated least squares problem with a modified righthand side; the modification is then updated using soft thresholding. With factorization caching, the cost of subsequent least squares iterations is much smaller than the initial one, often making the time required to carry out least absolute deviations very nearly the same as the time required to carry out least squares.

6.1.1 Huber Fitting

A problem that lies in between least squares and least absolute deviations is *Huber function fitting*,

$$\text{minimize} \quad g^{\text{hub}}(Ax - b),$$

where the *Huber penalty function* g^{hub} is quadratic for small arguments and transitions to an absolute value for larger values. For scalar a, it is given by

$$g^{\text{hub}}(a) = \begin{cases} a^2/2 & |a| \leq 1 \\ |a| - 1/2 & |a| > 1 \end{cases}$$

and extends to vector arguments as the sum of the Huber functions of the components. (For simplicity, we consider the standard Huber function, which transitions from quadratic to absolute value at the level 1.)

This can be put into ADMM form as above, and the ADMM algorithm is the same except that the z-update involves the proximity operator of the Huber function rather than that of the ℓ_1 norm:

$$z^{k+1} := \frac{\rho}{1+\rho}\left(Ax^{k+1} - b + u^k\right) + \frac{1}{1+\rho} S_{1+1/\rho}(Ax^{k+1} - b + u^k).$$

When the least squares fit $x^{\text{ls}} = (A^T A)^{-1} b$ satisfies $|x_i^{\text{ls}}| \leq 1$ for all i, it is also the Huber fit. In this case, ADMM terminates in two steps.

6.2 Basis Pursuit

Basis pursuit is the equality-constrained ℓ_1 minimization problem

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = b, \end{array}$$

with variable $x \in \mathbf{R}^n$, data $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, with $m < n$. Basis pursuit is often used as a heuristic for finding a sparse solution to an underdetermined system of linear equations. It plays a central role in modern statistical signal processing, particularly the theory of compressed sensing; see [24] for a recent survey.

In ADMM form, basis pursuit can be written as

$$\begin{array}{ll} \text{minimize} & f(x) + \|z\|_1 \\ \text{subject to} & x - z = 0, \end{array}$$

where f is the indicator function of $\{x \in \mathbf{R}^n \mid Ax = b\}$. The ADMM algorithm is then

$$\begin{aligned} x^{k+1} &:= \Pi(z^k - u^k) \\ z^{k+1} &:= S_{1/\rho}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}, \end{aligned}$$

where Π is projection onto $\{x \in \mathbf{R}^n \mid Ax = b\}$. The x-update, which involves solving a linearly-constrained minimum Euclidean norm problem, can be written explicitly as

$$x^{k+1} := (I - A^T(AA^T)^{-1}A)(z^k - u^k) + A^T(AA^T)^{-1}b.$$

Again, the comments on efficient computation from §4.2 apply; by caching a factorization of AA^T, subsequent projections are much cheaper than the first one. We can interpret ADMM for basis pursuit as reducing the solution of a least ℓ_1 norm problem to solving a sequence of minimum Euclidean norm problems. For a discussion of similar algorithms for related problems in image processing, see [2].

A recent class of algorithms called *Bregman iterative methods* have attracted considerable interest for solving ℓ_1 problems like basis pursuit. For basis pursuit and related problems, *Bregman iterative regularization* [176] is equivalent to the method of multipliers, and the *split Bregman method* [88] is equivalent to ADMM [68].

6.3 General ℓ_1 Regularized Loss Minimization

Consider the generic problem

$$\text{minimize} \quad l(x) + \lambda\|x\|_1, \tag{6.1}$$

where l is any convex loss function.

In ADMM form, this problem can be written as

$$\begin{array}{ll} \text{minimize} & l(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array}$$

where $g(z) = \lambda\|z\|_1$. The algorithm is

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(l(x) + (\rho/2)\|x - z^k + u^k\|_2^2 \right) \\ z^{k+1} &:= S_{\lambda/\rho}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

The x-update is a proximal operator evaluation. If l is smooth, this can be done by any standard method, such as Newton's method, a quasi-Newton method such as L-BFGS, or the conjugate gradient method. If l is quadratic, the x-minimization can be carried out by solving linear equations, as in §4.2. In general, we can interpret ADMM for ℓ_1 regularized loss minimization as reducing it to solving a sequence of ℓ_2 (squared) regularized loss minimization problems.

A very wide variety of models can be represented with the loss function l, including generalized linear models [122] and generalized additive models [94]. In particular, generalized linear models subsume linear regression, logistic regression, softmax regression, and Poisson regression, since they allow for any exponential family distribution. For general background on models like ℓ_1 regularized logistic regression, see, *e.g.*, [95, §4.4.4].

In order to use a regularizer $g(z)$ other than $\|z\|_1$, we simply replace the soft thresholding operator in the z-update with the proximity operator of g, as in §4.1.

6.4 Lasso

An important special case of (6.1) is ℓ_1 regularized linear regression, also called the *lasso* [156]. This involves solving

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1, \tag{6.2}$$

where $\lambda > 0$ is a scalar regularization parameter that is usually chosen by cross-validation. In typical applications, there are many more features than training examples, and the goal is to find a parsimonious model for the data. For general background on the lasso, see [95, §3.4.2]. The lasso has been widely applied, particularly in the analysis of biological data, where only a small fraction of a huge number of possible factors are actually predictive of some outcome of interest; see [95, §18.4] for a representative case study.

In ADMM form, the lasso problem can be written as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array}$$

where $f(x) = (1/2)\|Ax - b\|_2^2$ and $g(z) = \lambda\|z\|_1$. By §4.2 and §4.4, ADMM becomes

$$\begin{aligned} x^{k+1} &:= (A^TA + \rho I)^{-1}(A^Tb + \rho(z^k - u^k)) \\ z^{k+1} &:= S_{\lambda/\rho}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

Note that $A^TA + \rho I$ is always invertible, since $\rho > 0$. The x-update is essentially a *ridge regression* (*i.e.*, quadratically regularized least squares) computation, so ADMM can be interpreted as a method for solving the lasso problem by iteratively carrying out ridge regression. When using a direct method, we can cache an initial factorization to make subsequent iterations much cheaper. See [1] for an example of an application in image processing.

6.4.1 Generalized Lasso

The lasso problem can be generalized to

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|Fx\|_1, \tag{6.3}$$

where F is an arbitrary linear transformation. An important special case is when $F \in \mathbf{R}^{(n-1)\times n}$ is the difference matrix,

$$F_{ij} = \begin{cases} 1 & j = i+1 \\ -1 & j = i \\ 0 & \text{otherwise,} \end{cases}$$

and $A = I$, in which case the generalization reduces to

$$\text{minimize} \quad (1/2)\|x-b\|_2^2 + \lambda \textstyle\sum_{i=1}^{n-1} |x_{i+1} - x_i|. \tag{6.4}$$

The second term is the *total variation* of x. This problem is often called *total variation denoising* [145], and has applications in signal processing. When $A = I$ and F is a second difference matrix, the problem (6.3) is called ℓ_1 *trend filtering* [101].

In ADMM form, the problem (6.3) can be written as

$$\begin{array}{ll} \text{minimize} & (1/2)\|Ax-b\|_2^2 + \lambda\|z\|_1 \\ \text{subject to} & Fx - z = 0, \end{array}$$

which yields the ADMM algorithm

$$\begin{aligned} x^{k+1} &:= (A^TA + \rho F^TF)^{-1}(A^Tb + \rho F^T(z^k - u^k)) \\ z^{k+1} &:= S_{\lambda/\rho}(Fx^{k+1} + u^k) \\ u^{k+1} &:= u^k + Fx^{k+1} - z^{k+1}. \end{aligned}$$

For the special case of total variation denoising (6.4), $A^TA + \rho F^TF$ is tridiagonal, so the x-update can be carried out in $O(n)$ flops [90, §4.3]. For ℓ_1 trend filtering, the matrix is pentadiagonal, so the x-update is still $O(n)$ flops.

6.4.2 Group Lasso

As another example, consider replacing the regularizer $\|x\|_1$ with $\sum_{i=1}^N \|x_i\|_2$, where $x = (x_1, \ldots, x_N)$, with $x_i \in \mathbf{R}^{n_i}$. When $n_i = 1$ and $N = n$, this reduces to the ℓ_1 regularized problem (6.1). Here the regularizer is separable with respect to the partition $x_1, \ldots, x_N$ but not fully separable. This extension of ℓ_1 norm regularization is called the *group lasso* [177] or, more generally, *sum-of-norms regularization* [136].

ADMM for this problem is the same as above with the z-update replaced with block soft thresholding

$$z_i^{k+1} = \mathcal{S}_{\lambda/\rho}(x_i^{k+1} + u^k), \quad i = 1, \ldots, N,$$

where the vector soft thresholding operator $\mathcal{S}_\kappa : \mathbf{R}^m \to \mathbf{R}^m$ is

$$\mathcal{S}_\kappa(a) = (1 - \kappa/\|a\|_2)_+ a,$$

with $\mathcal{S}_\kappa(0) = 0$. This formula reduces to the scalar soft thresholding operator when a is a scalar, and generalizes the expression given in (4.2).

This can be extended further to handle overlapping groups, which is often useful in bioinformatics and other applications [181, 118]. In this case, we have N potentially overlapping groups $G_i \subseteq \{1, \ldots, n\}$ of variables, and the objective is

$$(1/2)\|Ax - b\|_2^2 + \lambda \sum_{i=1}^{N} \|x_{G_i}\|_2,$$

where x_{G_i} is the subvector of x with entries in G_i. Because the groups can overlap, this kind of objective is difficult to optimize with many standard methods, but it is straightforward with ADMM. To use ADMM, introduce N new variables $x_i \in \mathbf{R}^{|G_i|}$ and consider the problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|Az - b\|_2^2 + \lambda \sum_{i=1}^N \|x_i\|_2 \\ \text{subject to} & x_i - \tilde{z}_i = 0, \quad i = 1, \ldots, N, \end{array}$$

with local variables x_i and global variable z. Here, $\tilde{z}_i$ is the global variable z's idea of what the local variable x_i should be, and is given by a linear function of z. This follows the notation for general form consensus optimization outlined in full detail in §7.2; the overlapping group lasso problem above is a special case.

6.5 Sparse Inverse Covariance Selection

Given a dataset consisting of samples from a zero mean Gaussian distribution in $\mathbf{R}^n$,

$$a_i \sim \mathcal{N}(0, \Sigma), \quad i = 1, \ldots, N,$$

consider the task of estimating the covariance matrix Σ under the prior assumption that Σ^{-1} is sparse. Since $(\Sigma^{-1})_{ij}$ is zero if and only if the ith and jth components of the random variable are conditionally independent, given the other variables, this problem is equivalent to the *structure learning* problem of estimating the topology of the undirected graphical model representation of the Gaussian [104]. Determining the sparsity pattern of the inverse covariance matrix Σ^{-1} is also called the *covariance selection problem*.

For n very small, it is feasible to search over all sparsity patterns in Σ^{-1} since for a fixed sparsity pattern, determining the maximum likelihood estimate of Σ is a tractable (convex optimization) problem. A good heuristic that scales to much larger values of n is to minimize the negative log-likelihood (with respect to the parameter $X = \Sigma^{-1}$) with an ℓ_1 regularization term to promote sparsity of the estimated inverse covariance matrix [7]. If S is the empirical covariance matrix $(1/N)\sum_{i=1}^N a_i a_i^T$, then the estimation problem can be written as

$$\text{minimize} \quad \mathbf{Tr}(SX) - \log\det X + \lambda\|X\|_1,$$

with variable $X \in \mathbf{S}_+^n$, where $\|\cdot\|_1$ is defined elementwise, *i.e.*, as the sum of the absolute values of the entries, and the domain of $\log\det$ is $\mathbf{S}_{++}^n$, the set of symmetric positive definite $n \times n$ matrices. This is a special case of the general ℓ_1 regularized problem (6.1) with (convex) loss function $l(X) = \mathbf{Tr}(SX) - \log\det X$.

The idea of covariance selection is originally due to Dempster [48] and was first studied in the sparse, high-dimensional regime by Meinshausen and Bühlmann [123]. The form of the problem above is due to Banerjee et al. [7]. Some other recent papers on this problem include Friedman et al.'s *graphical lasso* [79], Duchi et al. [55], Lu [115], Yuan [178], and Scheinberg et al. [148], the last of which shows that ADMM outperforms state-of-the-art methods for this problem.

The ADMM algorithm for sparse inverse covariance selection is

$$\begin{aligned}
X^{k+1} &:= \operatorname*{argmin}_X \left(\mathbf{Tr}(SX) - \log\det X + (\rho/2)\|X - Z^k + U^k\|_F^2\right)\\
Z^{k+1} &:= \operatorname*{argmin}_Z \left(\lambda\|Z\|_1 + (\rho/2)\|X^{k+1} - Z + U^k\|_F^2\right)\\
U^{k+1} &:= U^k + X^{k+1} - Z^{k+1},
\end{aligned}$$

where $\|\cdot\|_F$ is the Frobenius norm, *i.e.*, the square root of the sum of the squares of the entries.

This algorithm can be simplified much further. The Z-minimization step is elementwise soft thresholding,

$$Z_{ij}^{k+1} := S_{\lambda/\rho}(X_{ij}^{k+1} + U_{ij}^k),$$

and the X-minimization also turns out to have an analytical solution. The first-order optimality condition is that the gradient should vanish,

$$S - X^{-1} + \rho(X - Z^k + U^k) = 0,$$

together with the implicit constraint $X \succ 0$. Rewriting, this is

$$\rho X - X^{-1} = \rho(Z^k - U^k) - S. \tag{6.5}$$

We will construct a matrix X that satisfies this condition and thus minimizes the X-minimization objective. First, take the orthogonal eigenvalue decomposition of the righthand side,

$$\rho(Z^k - U^k) - S = Q\Lambda Q^T,$$

where $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n)$, and $Q^TQ = QQ^T = I$. Multiplying (6.5) by Q^T on the left and by Q on the right gives

$$\rho\tilde{X} - \tilde{X}^{-1} = \Lambda,$$

where $\tilde{X} = Q^TXQ$. We can now construct a diagonal solution of this equation, *i.e.*, find positive numbers $\tilde{X}_{ii}$ that satisfy $\rho\tilde{X}_{ii} - 1/\tilde{X}_{ii} = \lambda_i$. By the quadratic formula,

$$\tilde{X}_{ii} = \frac{\lambda_i + \sqrt{\lambda_i^2 + 4\rho}}{2\rho},$$

which are always positive since $\rho > 0$. It follows that $X = Q\tilde{X}Q^T$ satisfies the optimality condition (6.5), so this is the solution to the X-minimization. The computational effort of the X-update is that of an eigenvalue decomposition of a symmetric matrix.

7

Consensus and Sharing

Here we describe two generic optimization problems, *consensus* and *sharing*, and ADMM-based methods for solving them using distributed optimization. Consensus problems have a long history, especially in conjunction with ADMM; see, *e.g.*, Bertsekas and Tsitsiklis [17] for a discussion of distributed consensus problems in the context of ADMM from the 1980s. Some more recent examples include a survey by Nedić and Ozdaglar [131] and several application papers by Giannakis and co-authors in the context of signal processing and wireless communications, such as [150, 182, 121].

7.1 Global Variable Consensus Optimization

We first consider the case with a single global variable, with the objective and constraint terms split into N parts:

$$\text{minimize} \quad f(x) = \sum_{i=1}^{N} f_i(x),$$

where $x \in \mathbf{R}^n$, and $f_i : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are convex. We refer to f_i as the ith term in the objective. Each term can also encode constraints by assigning $f_i(x) = +\infty$ when a constraint is violated. The goal is to

solve the problem above in such a way that each term can be handled by its own processing element, such as a thread or processor.

This problem arises in many contexts. In model fitting, for example, x represents the parameters in a model and f_i represents the loss function associated with the ith block of data or measurements. In this case, we would say that x is found by *collaborative filtering*, since the data sources are 'collaborating' to develop a global model.

This problem can be rewritten with local variables $x_i \in \mathbf{R}^n$ and a common global variable z:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N. \end{array} \tag{7.1}$$

This is called the *global consensus problem*, since the constraint is that all the local variables should agree, *i.e.*, be equal. Consensus can be viewed as a simple technique for turning additive objectives $\sum_{i=1}^N f_i(x)$, which show up frequently but do not split due to the variable being shared across terms, into separable objectives $\sum_{i=1}^N f_i(x_i)$, which split easily. For a useful recent discussion of consensus algorithms, see [131] and the references therein.

ADMM for the problem (7.1) can be derived either directly from the augmented Lagrangian

$$L_\rho(x_1, \ldots, x_N, z, y) = \sum_{i=1}^N \left(f_i(x_i) + y_i^T(x_i - z) + (\rho/2)\|x_i - z\|_2^2 \right),$$

or simply as a special case of the constrained optimization problem (5.1) with variable $(x_1, \ldots, x_N) \in \mathbf{R}^{nN}$ and constraint set

$$\mathcal{C} = \{(x_1, \ldots, x_N) \mid x_1 = x_2 = \cdots = x_N\}.$$

The resulting ADMM algorithm is the following:

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)\|x_i - z^k\|_2^2 \right) \\ z^{k+1} &:= \frac{1}{N} \sum_{i=1}^N \left(x_i^{k+1} + (1/\rho) y_i^k \right) \\ y_i^{k+1} &:= y_i^k + \rho(x_i^{k+1} - z^{k+1}). \end{aligned}$$

Here, we write y^{kT} instead of $(y^k)^T$ to lighten the notation. The first and last steps are carried out independently for each $i = 1, \ldots, N$. In the literature, the processing element that handles the global variable z is sometimes called the *central collector* or the *fusion center*. Note that the z-update is simply the projection of $x^{k+1} + (1/\rho)y^k$ onto the constraint set $\mathcal{C}$ of 'block-constant' vectors.

This algorithm can be simplified further. With the average (over $i = 1, \ldots, N$) of a vector denoted with an overline, the z-update can be written

$$z^{k+1} = \overline{x}^{k+1} + (1/\rho)\overline{y}^k.$$

Similarly, averaging the y-update gives

$$\overline{y}^{k+1} = \overline{y}^k + \rho(\overline{x}^{k+1} - z^{k+1}).$$

Substituting the first equation into the second shows that $\overline{y}^{k+1} = 0$, *i.e.*, the dual variables have average value zero after the first iteration. Using $z^k = \overline{x}^k$, ADMM can be written as

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT}(x_i - \overline{x}^k) + (\rho/2)\|x_i - \overline{x}^k\|_2^2 \right) \\ y_i^{k+1} &:= y_i^k + \rho(x_i^{k+1} - \overline{x}^{k+1}). \end{aligned}$$

We have already seen a special case of this in parallel projections (see §5.1.2), which is consensus ADMM for the case when f_i are all indicator functions of convex sets.

This is a very intuitive algorithm. The dual variables are separately updated to drive the variables into consensus, and quadratic regularization helps pull the variables toward their average value while still attempting to minimize each local f_i.

We can interpret consensus ADMM as a method for solving problems in which the objective and constraints are distributed across multiple processors. Each processor only has to handle its own objective and constraint term, plus a quadratic term which is updated each iteration. The quadratic terms (or more accurately, the linear parts of the quadratic terms) are updated in such a way that the variables converge to a common value, which is the solution of the full problem.

For consensus ADMM, the primal and dual residuals are

$$r^k = (x_1^k - \overline{x}^k, \ldots, x_N^k - \overline{x}^k), \qquad s^k = -\rho(\overline{x}^k - \overline{x}^{k-1}, \ldots, \overline{x}^k - \overline{x}^{k-1}),$$

so their (squared) norms are

$$\|r^k\|_2^2 = \sum_{i=1}^N \|x_i^k - \overline{x}^k\|_2^2, \qquad \|s^k\|_2^2 = N\rho^2\|\overline{x}^k - \overline{x}^{k-1}\|_2^2.$$

The first term is N times the standard deviation of the points $x_1, \ldots, x_N$, a natural measure of (lack of) consensus.

When the original consensus problem is a parameter fitting problem, the x-update step has an intuitive statistical interpretation. Suppose f_i is the negative log-likelihood function for the parameter x, given the measurements or data available to the ith processing element. Then x_i^{k+1} is precisely the maximum a posteriori (MAP) estimate of the parameter, given the Gaussian prior distribution $\mathcal{N}(\overline{x}^k + (1/\rho)y_i^k, \rho I)$. The expression for the prior mean is also intuitive: It is the average value $\overline{x}^k$ of the local parameter estimates in the previous iteration, translated slightly by y_i^k, the 'price' of the ith processor disagreeing with the consensus in the previous iteration. Note also that the use of different forms of penalty in the augmented term, as discussed in §3.4, will lead to corresponding changes in this prior distribution; for example, using a matrix penalty P rather than a scalar ρ will mean that the Gaussian prior distribution has covariance P rather than ρI.

7.1.1 Global Variable Consensus with Regularization

In a simple variation on the global variable consensus problem, an objective term g, often representing a simple constraint or regularization, is handled by the central collector:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + g(z) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N. \end{array} \tag{7.2}$$

The resulting ADMM algorithm is

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)\|x_i - z^k\|_2^2 \right) \quad (7.3)$$

$$z^{k+1} := \operatorname*{argmin}_{z} \left(g(z) + \sum_{i=1}^{N} (-y_i^{kT} z + (\rho/2)\|x_i^{k+1} - z\|_2^2) \right) \quad (7.4)$$

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - z^{k+1}). \quad (7.5)$$

By collecting the linear and quadratic terms, we can express the z-update as an averaging step, as in consensus ADMM, followed by a proximal step involving g:

$$z^{k+1} := \operatorname*{argmin}_{z} \left(g(z) + (N\rho/2)\|z - \overline{x}^{k+1} - (1/\rho)\overline{y}^k\|_2^2 \right).$$

In the case with nonzero g, we do not in general have $\overline{y}^k = 0$, so we cannot drop the y_i terms from z-update as in consensus ADMM.

As an example, for $g(z) = \lambda\|z\|_1$, with $\lambda > 0$, the second step of the z-update is a soft threshold operation:

$$z^{k+1} := S_{\lambda/N\rho}(\overline{x}^{k+1} - (1/\rho)\overline{y}^k).$$

As another simple example, suppose g is the indicator function of $\mathbf{R}_+^n$, which means that the g term enforces nonnegativity of the variable. In this case, the update is

$$z^{k+1} := (\overline{x}^{k+1} - (1/\rho)\overline{y}^k)_+.$$

The scaled form of ADMM for this problem also has an appealing form, which we record here for convenience:

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right) \quad (7.6)$$

$$z^{k+1} := \operatorname*{argmin}_{z} \left(g(z) + (N\rho/2)\|z - \overline{x}^{k+1} - \overline{u}^k\|_2^2 \right) \quad (7.7)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}. \quad (7.8)$$

In many cases, this version is simpler and easier to work with than the unscaled form.

7.2 General Form Consensus Optimization

We now consider a more general form of the consensus minimization problem, in which we have local variables $x_i \in \mathbf{R}^{n_i}$, $i = 1, \ldots, N$, with the objective $f_1(x_1) + \cdots + f_N(x_N)$ separable in the x_i. Each of these local variables consists of a selection of the components of the global variable $z \in \mathbf{R}^n$; that is, each component of each local variable corresponds to some global variable component z_g. The mapping from local variable indices into global variable index can be written as $g = \mathcal{G}(i, j)$, which means that local variable component $(x_i)_j$ corresponds to global variable component z_g.

Achieving consensus between the local variables and the global variable means that

$$(x_i)_j = z_{\mathcal{G}(i,j)}, \quad i = 1, \ldots, N, \quad j = 1, \ldots, n_i.$$

If $\mathcal{G}(i, j) = j$ for all i, then each local variable is just a copy of the global variable, and consensus reduces to global variable consensus, $x_i = z$. General consensus is of interest in cases where $n_i \ll n$, so each local vector only contains a small number of the global variables.

In the context of model fitting, the following is one way that general form consensus naturally arises. The global variable z is the full feature vector (*i.e.*, vector of model parameters or independent variables in the data), and different subsets of the data are spread out among N processors. Then x_i can be viewed as the subvector of z corresponding to (nonzero) features that appear in the ith block of data. In other words, each processor handles only its block of data *and* only the subset of model coefficients that are relevant for that block of data. If in each block of data all regressors appear with nonzero values, then this reduces to global consensus.

For example, if each training example is a document, then the features may include words or combinations of words in the document; it will often be the case that some words are only used in a small subset of the documents, in which case each processor can just deal with the words that appear in its local corpus. In general, datasets that are high-dimensional but sparse will benefit from this approach.

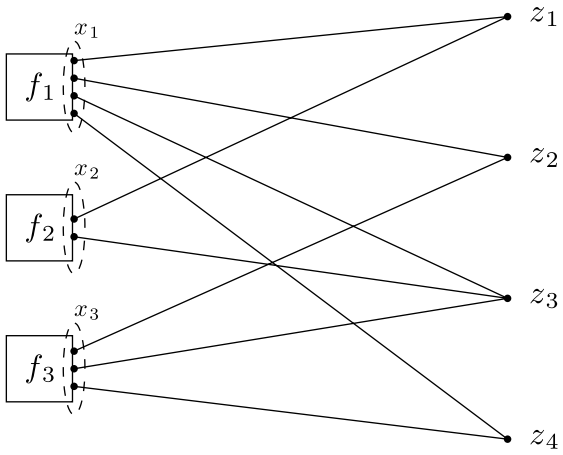

Fig. 7.1. General form consensus optimization. Local objective terms are on the left; global variable components are on the right. Each edge in the bipartite graph is a consistency constraint, linking a local variable and a global variable component.

For ease of notation, let $\tilde{z}_i \in \mathbf{R}^{n_i}$ be defined by $(\tilde{z}_i)_j = z_{\mathcal{G}(i,j)}$. Intuitively, $\tilde{z}_i$ is the global variable's idea of what the local variable x_i should be; the consensus constraint can then be written very simply as $x_i - \tilde{z}_i = 0$, $i = 1, \ldots, N$.

The general form consensus problem is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_i - \tilde{z}_i = 0, \quad i = 1, \ldots, N, \end{array} \tag{7.9}$$

with variables $x_1, \ldots, x_N$ and z ($\tilde{z}_i$ are linear functions of z).

A simple example is shown in Figure 7.1. In this example, we have $N = 3$ subsystems, global variable dimension $n = 4$, and local variable dimensions $n_1 = 4$, $n_2 = 2$, and $n_3 = 3$. The objective terms and global variables form a bipartite graph, with each edge representing a consensus constraint between a local variable component and a global variable.

The augmented Lagrangian for (7.9) is

$$L_\rho(x, z, y) = \sum_{i=1}^N \left(f_i(x_i) + y_i^T(x_i - \tilde{z}_i) + (\rho/2)\|x_i - \tilde{z}_i\|_2^2 \right),$$

with dual variable $y_i \in \mathbf{R}^{n_i}$. Then ADMM consists of the iterations

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT} x_i + (\rho/2)\|x_i - \tilde{z}_i^k\|_2^2 \right) \\ z^{k+1} &:= \operatorname*{argmin}_{z} \left(\sum_{i=1}^{m} \left(-y_i^{kT} \tilde{z}_i + (\rho/2)\|x_i^{k+1} - \tilde{z}_i\|_2^2 \right) \right) \\ y_i^{k+1} &:= y_i^k + \rho(x_i^{k+1} - \tilde{z}_i^{k+1}), \end{aligned}$$

where the x_i- and y_i-updates can be carried out independently in parallel for each i.

The z-update step decouples across the components of z, since L_ρ is fully separable in its components:

$$z_g^{k+1} := \frac{\sum_{\mathcal{G}(i,j)=g} \left((x_i^{k+1})_j + (1/\rho)(y_i^k)_j \right)}{\sum_{\mathcal{G}(i,j)=g} 1},$$

so z_g is found by averaging all entries of $x_i^{k+1} + (1/\rho)y_i^k$ that correspond to the global index g. Applying the same type of argument as in the global variable consensus case, we can show that after the first iteration,

$$\sum_{\mathcal{G}(i,j)=g} (y_i^k)_j = 0,$$

i.e., the sum of the dual variable entries that correspond to any given global index g is zero. The z-update step can thus be written in the simpler form

$$z_g^{k+1} := (1/k_g) \sum_{\mathcal{G}(i,j)=g} (x_i^{k+1})_j,$$

where k_g is the number of local variable entries that correspond to global variable entry z_g. In other words, the z-update is local averaging for each component z_g rather than global averaging; in the language of collaborative filtering, we could say that only the processing elements that have an opinion on a feature z_g will vote on z_g.

7.2.1 General Form Consensus with Regularization

As in the global consensus case, the general form consensus problem can be generalized by allowing the global variable nodes to handle an

objective term. Consider the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + g(z) \\ \text{subject to} & x_i - \tilde{z}_i = 0, \quad i = 1, \ldots, N, \end{array} \tag{7.10}$$

where g is a regularization function. The z-update involves the local averaging step from the unregularized setting, followed by an application of the proximity operator $\mathbf{prox}_{g,k_g\rho}$ to the results of this averaging, just as in the global variable consensus case.

7.3 Sharing

Another canonical problem that will prove useful in the sequel is the *sharing problem*

$$\text{minimize} \quad \sum_{i=1}^N f_i(x_i) + g\left(\sum_{i=1}^N x_i\right) \tag{7.11}$$

with variables $x_i \in \mathbf{R}^n$, $i = 1, \ldots, N$, where f_i is a local cost function for subsystem i, and g is the shared objective, which takes as argument the sum of the variables. We can think of the variable x_i as being the choice of agent i; the sharing problem involves each agent adjusting its variable to minimize its individual cost $f_i(x_i)$, as well as the shared objective term $g(\sum_{i=1}^N x_i)$. The sharing problem is important both because many useful problems can be put into this form and because it enjoys a dual relationship with the consensus problem, as discussed below.

Sharing can be written in ADMM form by copying all the variables:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + g\left(\sum_{i=1}^N z_i\right) \\ \text{subject to} & x_i - z_i = 0, \quad i = 1, \ldots, N, \end{array} \tag{7.12}$$

with variables $x_i,\ z_i \in \mathbf{R}^n$, $i = 1, \ldots, N$. The scaled form of ADMM is

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\rho/2)\|x_i - z_i^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= \operatorname*{argmin}_{z} \left(g\left(\textstyle\sum_{i=1}^N z_i\right) + (\rho/2) \textstyle\sum_{i=1}^N \|z_i - u_i^k - x_i^{k+1}\|_2^2 \right) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z_i^{k+1}. \end{aligned}$$

The first and last steps can be carried out independently in parallel for each $i = 1, \ldots, N$. As written, the z-update requires solving a problem

in Nn variables, but we will show that it is possible to carry it out by solving a problem in only n variables.

For simplicity of notation, let $a_i = u_i^k + x_i^{k+1}$. Then the z-update can be rewritten as

$$\begin{array}{ll} \text{minimize} & g(N\overline{z}) + (\rho/2)\sum_{i=1}^N \|z_i - a_i\|_2^2 \\ \text{subject to} & \overline{z} = (1/N)\sum_{i=1}^N z_i, \end{array}$$

with additional variable $\overline{z} \in \mathbf{R}^n$. Minimizing over $z_1, \ldots, z_N$ with $\overline{z}$ fixed has the solution

$$z_i = a_i + \overline{z} - \overline{a}, \tag{7.13}$$

so the z-update can be computed by solving the unconstrained problem

$$\text{minimize} \quad g(N\overline{z}) + (\rho/2)\sum_{i=1}^N \|\overline{z} - \overline{a}\|_2^2$$

for $\overline{z} \in \mathbf{R}^n$ and then applying (7.13). Substituting (7.13) for z_i^{k+1} in the u-update gives

$$u_i^{k+1} = \overline{u}^k + \overline{x}^{k+1} - \overline{z}^{k+1}, \tag{7.14}$$

which shows that the dual variables u_i^k are all equal (*i.e.*, in consensus) and can be replaced with a single dual variable $u \in \mathbf{R}^m$. Substituting in the expression for z_i^k in the x-update, the final algorithm becomes

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\rho/2)\|x_i - x_i^k + \overline{x}^k - \overline{z}^k + u^k\|_2^2 \right) \\ \overline{z}^{k+1} &:= \operatorname*{argmin}_{\overline{z}} \left(g(N\overline{z}) + (N\rho/2)\|\overline{z} - u^k - \overline{x}^{k+1}\|_2^2 \right) \\ u^{k+1} &:= u^k + \overline{x}^{k+1} - \overline{z}^{k+1}. \end{aligned}$$

The x-update can be carried out in parallel, for $i = 1, \ldots, N$. The z-update step requires gathering x_i^{k+1} to form the averages, and then solving a problem with n variables. After the u-update, the new value of $\overline{x}^{k+1} - \overline{z}^{k+1} + u^{k+1}$ is scattered to the subsystems.

7.3.1 Duality

Attaching Lagrange multipliers ν_i to the constraints $x_i - z_i = 0$, the dual function Γ of the ADMM sharing problem (7.12) is given by

$$\Gamma(\nu_1, \ldots, \nu_N) = \begin{cases} -g^*(\nu_1) - \sum_i f_i^*(-\nu_i) & \text{if } \nu_1 = \nu_2 = \cdots = \nu_N \\ -\infty & \text{otherwise.} \end{cases}$$

Letting $\psi = g^*$ and $h_i(\nu) = f_i^*(-\nu)$, the dual sharing problem can be written as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N h_i(\nu_i) + \psi(\nu) \\ \text{subject to} & \nu_i - \nu = 0, \end{array} \tag{7.15}$$

with variables $\nu \in \mathbf{R}^n$, $\nu_i \in \mathbf{R}^n$, $i = 1, \ldots, N$. This is identical to the regularized global variable consensus problem (7.2). Assuming strong duality holds, this implies that $y^k = \rho u^k \to \nu^\star$ in ADMM, where $\nu^\star$ is an optimal point of (7.15).

Consider the reverse direction. Attaching Lagrange multipliers $d_i \in \mathbf{R}^n$ to the constraints $\nu_i - \nu = 0$, the dual of the regularized global consensus problem is

$$\text{minimize} \quad \sum_{i=1}^N f_i(d_i) + g(\sum_{i=1}^N d_i)$$

with variables $d_i \in \mathbf{R}^n$, which is exactly the sharing problem (7.11). (This follows because f and g are assumed to be convex and closed, so $f^{**} = f$ and $g^{**} = g$.) Assuming strong duality holds, running ADMM on the consensus problem (7.15) gives that $d_i^k \to x_i^\star$, where $x_i^\star$ is an optimal point of the sharing problem (7.11).

Thus, there is a close dual relationship between the consensus problem (7.15) and the sharing problem (7.11). In fact, the global consensus problem can be solved by running ADMM on its dual sharing problem, and vice versa. This is related to work by Fukushima [80] on 'dual ADMM' methods.

7.3.2 Optimal Exchange

Here, we highlight an important special case of the sharing problem with an appealing economic interpretation. The *exchange problem* is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & \sum_{i=1}^N x_i = 0, \end{array} \tag{7.16}$$

with variables $x_i \in \mathbf{R}^n$, $i = 1, \ldots, N$, where f_i represents the cost function for subsystem i. This is a sharing problem where the shared objective g is the indicator function of the set $\{0\}$. The components of the vectors x_i represent quantities of commodities that are exchanged

among N agents or subsystems. When $(x_i)_j$ is nonnegative, it can be viewed as the amount of commodity j *received* by subsystem i from the exchange. When $(x_i)_j$ is negative, its magnitude $|(x_i)_j|$ can be viewed as the amount of commodity j *contributed* by subsystem i to the exchange. The *equilibrium constraint* that each commodity clears, or balances, is simply $\sum_{i=1}^N x_i = 0$. As this interpretation suggests, this and related problems have a long history in economics, particularly in the theories of market exchange, resource allocation, and general equilibrium; see, for example, the classic works by Walras [168], Arrow and Debreu [4], and Uzawa [162, 163].

The exchange problem can be solved via ADMM either by applying the generic sharing algorithm above and simplifying, or by treating it as a generic constrained convex problem (5.1), with

$$\mathcal{C} = \left\{x \in \mathbf{R}^{nN} \mid x_1 + \cdots + x_N = 0\right\}.$$

This gives the exchange ADMM algorithm

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\rho/2)\|x_i - x_i^k + \overline{x}^k + u^k\|_2^2\right) \\ u^{k+1} &:= u^k + \overline{x}^{k+1}. \end{aligned}$$

It is also instructive to consider the unscaled form of ADMM for this problem:

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y^{kT} x_i + (\rho/2)\|x_i - (x_i^k - \overline{x}^k)\|_2^2\right) \\ y^{k+1} &:= y^k + \rho\overline{x}^{k+1}. \end{aligned}$$

The variable y^k converges to an optimal dual variable, which is readily interpreted as a set of optimal or clearing prices for the exchange. The proximal term in the x-update is a penalty for x^{k+1} deviating from x^k, projected onto the feasible set. The x-update in exchange ADMM can be carried out independently in parallel, for $i = 1, \ldots, N$. The u-update requires gathering the x_i^{k+1} (or otherwise averaging), and broadcasting $\overline{x}^{k+1} + u^{k+1}$ back to the processors handling the x_i updates.

Exchange ADMM can be viewed as a form of *tâtonnement* or *price adjustment* process [168, 163] from Walras' theory of general equilibrium. Tâtonnement represents the mechanism of the competitive market working towards market equilibrium; the idea is that the market

acts via price adjustment, *i.e.*, increasing or decreasing the price of each good depending on whether there is an excess demand or excess supply of the good, respectively.

Dual decomposition is the simplest algorithmic expression of tâtonnement. In this setting, each agent adjusts his consumption x_i to minimize his individual cost $f_i(x_i)$ adjusted by the cost $y^T x_i$, where y is the price vector. The central collector (called the 'secretary of market' in [163]) works toward equilibrium by adjusting the prices y up or down depending on whether each commodity or good is overproduced or underproduced. ADMM differs only in the inclusion of the proximal regularization term in the updates for each agent. As y^k converges to an optimal price vector $y^\star$, the effect of the proximal regularization term vanishes. The proximal regularization term can be interpreted as each agent's commitment to help clear the market.

8

Distributed Model Fitting

A general convex model fitting problem can be written in the form

$$\text{minimize} \quad l(Ax - b) + r(x), \tag{8.1}$$

with parameters $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m \times n}$ is the feature matrix, $b \in \mathbf{R}^m$ is the output vector, $l : \mathbf{R}^m \to \mathbf{R}$ is a convex loss function, and r is a convex regularization function. We assume that l is additive, so

$$l(Ax - b) = \sum_{i=1}^{m} l_i(a_i^T x - b_i),$$

where $l_i : \mathbf{R} \to \mathbf{R}$ is the loss for the ith training example, $a_i \in \mathbf{R}^n$ is the feature vector for example i, and b_i is the output or response for example i. Each l_i can be different, though in practice they are usually all the same.

We also assume that the regularization function r is separable. The most common examples are $r(x) = \lambda \|x\|_2^2$ (called *Tikhonov regularization*, or a *ridge penalty* in statistical settings) and $r(x) = \lambda \|x\|_1$ (sometimes generically called a *lasso penalty* in statistical settings), where λ is a positive regularization parameter, though more elaborate regularizers can be used just as easily. In some cases, one or more model

parameters are not regularized, such as the offset parameter in a classification model. This corresponds to, for example, $r(x) = \lambda\|x_{1:n-1}\|_1$, where $x_{1:n-1}$ is the subvector of x consisting of all but the last component of x; with this choice of r, the last component of x is not regularized.

The next section discusses some examples that have the general form above. We then consider two ways to solve (8.1) in a distributed manner, namely, by splitting across training examples and by splitting across features. While we work with the assumption that l and r are separable at the component level, we will see that the methods we describe work with appropriate block separability as well.

8.1 Examples

8.1.1 Regression

Consider a linear modeling problem with measurements of the form

$$b_i = a_i^T x + v_i,$$

where a_i is the ith feature vector and the measurement noises v_i are independent with log-concave densities p_i; see, *e.g.*, [20, §7.1.1]. Then the negative log-likelihood function is $l(Ax - b)$, with $l_i(\omega) = -\log p_i(-\omega)$. If $r = 0$, then the general fitting problem (8.1) can be interpreted as maximum likelihood estimation of x under noise model p_i. If r_i is taken to be the negative log prior density of x_i, then the problem can be interpreted as MAP estimation.

For example, the lasso follows the form above with quadratic loss $l(u) = (1/2)\|u\|_2^2$ and ℓ_1 regularization $r(x) = \lambda\|x\|_1$, which is equivalent to MAP estimation of a linear model with Gaussian noise and a Laplacian prior on the parameters [156, §5].

8.1.2 Classification

Many classification problems can also be put in the form of the general model fitting problem (8.1), with A, b, l, and r appropriately chosen. We follow the standard setup from statistical learning theory, as described in, *e.g.*, [8]. Let $p_i \in \mathbf{R}^{n-1}$ denote the feature vector of the ith example

and let $q_i \in \{-1, 1\}$ denote the binary outcome or class label, for $i = 1, \ldots, m$. The goal is to find a weight vector $w \in \mathbf{R}^{n-1}$ and offset $v \in \mathbf{R}$ such that

$$\mathbf{sign}(p_i^T w + v) = q_i$$

holds for many examples. Viewed as a function of p_i, the expression $p_i^T w + v$ is called a *discriminant function*. The condition that the sign of the discriminant function and the response should agree can also be written as $\mu_i > 0$, where $\mu_i = q_i(p_i^T w + v)$ is called the *margin* of the ith training example.

In the context of classification, loss functions are generally written as a function of the margin, so the loss for the ith example is

$$l_i(\mu_i) = l_i(q_i(p_i^T w + v)).$$

A classification error is made if and only if the margin is negative, so l_i should be positive and decreasing for negative arguments and zero or small for positive arguments. To find the parameters w and v, we minimize the average loss plus a regularization term on the weights:

$$\frac{1}{m} \sum_{i=1}^{m} l_i(q_i(p_i^T w + v)) + r^{\mathrm{wt}}(w). \tag{8.2}$$

This has the generic model fitting form (8.1), with $x = (w, v)$, $a_i = (q_i p_i, -q_i)$, $b_i = 0$, and regularizer $r(x) = r^{\mathrm{wt}}(w)$. (We also need to scale l_i by $1/m$.) In the sequel, we will address such problems using the form (8.1) without comment, assuming that this transformation has been carried out.

In statistical learning theory, the problem (8.2) is referred to as *penalized empirical risk minimization* or *structural risk minimization*. When the loss function is convex, this is sometimes termed *convex risk minimization*. In general, fitting a classifier by minimizing a *surrogate loss function*, *i.e.*, a convex upper bound to 0-1 loss, is a well studied and widely used approach in machine learning; see, *e.g.*, [165, 180, 8].

Many classification models in machine learning correspond to different choices of loss function l_i and regularization or penalty r^{wt}.

Some common loss functions are *hinge loss* $(1-\mu_i)_+$, *exponential loss* $\exp(-\mu_i)$, and *logistic loss* $\log(1+\exp(-\mu_i))$; the most common regularizers are ℓ_1 and ℓ_2 (squared). The *support vector machine* (SVM) [151] corresponds to hinge loss with a quadratic penalty, while exponential loss yields *boosting* [78] and logistic loss yields logistic regression.

8.2 Splitting across Examples

Here we discuss how to solve the model fitting problem (8.1) with a modest number of features but a very large number of training examples. Most classical statistical estimation problems belong to this regime, with large volumes of relatively low-dimensional data. The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.

We partition A and b by rows,

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_N \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix},$$

with $A_i \in \mathbf{R}^{m_i \times n}$ and $b_i \in \mathbf{R}^{m_i}$, where $\sum_{i=1}^N m_i = m$. Thus, A_i and b_i represent the ith block of data and will be handled by the ith processor. We first put the model fitting problem in the consensus form

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N l_i(A_i x_i - b_i) + r(z) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N, \end{array} \tag{8.3}$$

with variables $x_i \in \mathbf{R}^n$ and $z \in \mathbf{R}^n$. Here, l_i refers (with some abuse of notation) to the loss function for the ith block of data. The problem can now be solved by applying the generic global variable consensus ADMM

algorithm described in §7.1, given here with scaled dual variable:

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(l_i(A_i x_i - b_i) + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= \operatorname*{argmin}_{z} \left(r(z) + (N\rho/2)\|z - \overline{x}^{k+1} - \overline{u}^k\|_2^2 \right) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

The first step, which consists of an ℓ_2-regularized model fitting problem, can be carried out in parallel for each data block. The second step requires gathering variables to form the average. The minimization in the second step can be carried out componentwise (and usually analytically) when r is assumed to be fully separable.

The algorithm described above only requires that the loss function l be separable across the blocks of data; the regularizer r does not need to be separable at all. (However, when r is not separable, the z-update may require the solution of a nontrivial optimization problem.)

8.2.1 Lasso

For the lasso, this yields the distributed algorithm

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left((1/2)\|A_i x_i - b_i\|_2^2 + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= S_{\lambda/\rho N}(\overline{x}^{k+1} + \overline{u}^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

Each x_i-update takes the form of a Tikhonov-regularized least squares (*i.e.*, ridge regression) problem, with analytical solution

$$x_i^{k+1} := (A_i^T A_i + \rho I)^{-1}(A_i^T b_i + \rho(z^k - u_i^k)).$$

The techniques from §4.2 apply: If a direct method is used, then the factorization of $A_i^T A_i + \rho I$ can be cached to speed up subsequent updates, and if $m_i < n$, then the matrix inversion lemma can be applied to let us factor the smaller matrix $A_i A_i^T + \rho I$ instead.

Comparing this distributed-data lasso algorithm with the serial version in §6.4, we see that the only difference is the collection and averaging steps, which couple the computations for the data blocks.

An ADMM-based distributed lasso algorithm is described in [121], with applications in signal processing and wireless communications.

8.2.2 Sparse Logistic Regression

Consider solving (8.1) with logistic loss functions l_i and ℓ_1 regularization. We ignore the intercept term for notational simplicity; the algorithm can be easily modified to incorporate an intercept. The ADMM algorithm is

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(l_i(A_i x_i) + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= S_{\lambda/\rho N}(\overline{x}^{k+1} + \overline{u}^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

This is identical to the distributed lasso algorithm, except for the x_i update, which here involves an ℓ_2 regularized logistic regression problem that can be efficiently solved by algorithms like L-BFGS.

8.2.3 Support Vector Machine

Using the notation of (8.1), the algorithm is

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(\mathbf{1}^T (A_i x_i + \mathbf{1})_+ + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= \frac{\rho}{(1/\lambda) + N\rho} (\overline{x}^{k+1} + \overline{u}^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

Each x_i-update essentially involves fitting a support vector machine to the local data A_i (with an offset in the quadratic regularization term), so this can be carried out efficiently using an existing SVM solver for serial problems.

The use of ADMM to train support vector machines in a distributed fashion was described in [74].

8.3 Splitting across Features

Now we consider the model fitting problem (8.1) with a modest number of examples and a large number of features. Statistical problems of this kind frequently arise in areas like natural language processing and bioinformatics, where there are often a large number of potential

explanatory variables for any given outcome. For example, the observations may be a corpus of documents, and the features could include all words and pairs of adjacent words (bigrams) that appear in each document. In bioinformatics, there are usually relatively few people in a given association study, but there can be a very large number of potential features relating to factors like observed DNA mutations in each individual. There are many examples in other areas as well, and the goal is to solve such problems in a distributed fashion with each processor handling a subset of the features. In this section, we show how this can be done by formulating it as a sharing problem from §7.3.

We partition the parameter vector x as $x = (x_1, \ldots, x_N)$, with $x_i \in \mathbf{R}^{n_i}$, where $\sum_{i=1}^N n_i = n$. Conformably partition the data matrix A as $A = [A_1 \cdots A_N]$, with $A_i \in \mathbf{R}^{m \times n_i}$, and the regularization function as $r(x) = \sum_{i=1}^N r_i(x_i)$. This implies that $Ax = \sum_{i=1}^N A_i x_i$, *i.e.*, $A_i x_i$ can be thought of as a 'partial' prediction of b using only the features referenced in x_i. The model fitting problem (8.1) becomes

$$\text{minimize} \quad l\left(\sum_{i=1}^N A_i x_i - b\right) + \sum_{i=1}^N r_i(x_i).$$

Following the approach used for the sharing problem (7.12), we express the problem as

$$\begin{array}{ll} \text{minimize} & l\left(\sum_{i=1}^N z_i - b\right) + \sum_{i=1}^N r_i(x_i) \\ \text{subject to} & A_i x_i - z_i = 0, \quad i = 1, \ldots, N, \end{array}$$

with new variables $z_i \in \mathbf{R}^m$. The derivation and simplification of ADMM also follows that for the sharing problem. The scaled form of ADMM is

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(r_i(x_i) + (\rho/2)\|A_i x_i - z_i^k + u_i^k\|_2^2 \right) \\ z^{k+1} &:= \operatorname*{argmin}_{z} \left(l(\sum_{i=1}^N z_i - b) + \sum_{i=1}^N (\rho/2)\|A_i x_i^{k+1} - z_i^k + u_i^k\|_2^2 \right) \\ u_i^{k+1} &:= u_i^k + A_i x_i^{k+1} - z_i^{k+1}. \end{aligned}$$

As in the discussion for the sharing problem, we carry out the z-update by first solving for the average $\overline{z}^{k+1}$:

$$\begin{aligned}
\overline{z}^{k+1} &:= \operatorname*{argmin}_{\overline{z}} \left(l(N\overline{z} - b) + (N\rho/2)\|\overline{z} - \overline{Ax}^{k+1} - \overline{u}^k\|_2^2 \right) \\
z_i^{k+1} &:= \overline{z}^{k+1} + A_i x_i^{k+1} + u_i^k - \overline{Ax}^{k+1} - \overline{u}^k,
\end{aligned}$$

where $\overline{Ax}^{k+1} = (1/N)\sum_{i=1}^N A_i x_i^{k+1}$. Substituting the last expression into the update for u_i, we find that

$$u_i^{k+1} = \overline{Ax}^{k+1} + \overline{u}^k - \overline{z}^{k+1},$$

which shows that, as in the sharing problem, all the dual variables are equal. Using a single dual variable $u^k \in \mathbf{R}^m$, and eliminating z_i, we arrive at the algorithm

$$\begin{aligned}
x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(r_i(x_i) + (\rho/2)\|A_i x_i - A_i x_i^k - \overline{z}^k + \overline{Ax}^k + u^k\|_2^2 \right) \\
\overline{z}^{k+1} &:= \operatorname*{argmin}_{\overline{z}} \left(l(N\overline{z} - b) + (N\rho/2)\|\overline{z} - \overline{Ax}^{k+1} - u^k\|_2^2 \right) \\
u^{k+1} &:= u^k + \overline{Ax}^{k+1} - \overline{z}^{k+1}.
\end{aligned}$$

The first step involves solving N parallel regularized least squares problems in n_i variables each. Between the first and second steps, we collect and sum the partial predictors $A_i x_i^{k+1}$ to form $\overline{Ax}^{k+1}$. The second step is a single minimization in m variables, a quadratically regularized loss minimization problem; the third step is a simple update in m variables.

This algorithm does *not* require l to be separable in the training examples, as assumed earlier. If l is separable, then the $\overline{z}$-update fully splits into m separate scalar optimization problems. Similarly, the regularizer r only needs to be separable at the level of the blocks of features. For example, if r is a sum-of-norms, as in §6.4.2, then it would be natural to have each subsystem handle a separate group.

8.3.1 Lasso

In this case, the algorithm above becomes

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left((\rho/2)\|A_i x_i - A_i x_i^k - \overline{z}^k + \overline{Ax}^k + u^k\|_2^2 + \lambda\|x_i\|_1 \right)$$
$$\overline{z}^{k+1} := \frac{1}{N+\rho}\left(b + \rho\overline{Ax}^{k+1} + \rho u^k \right)$$
$$u^{k+1} := u^k + \overline{Ax}^{k+1} - \overline{z}^{k+1}.$$

Each x_i-update is a lasso problem with n_i variables, which can be solved using any single processor lasso method.

In the x_i-updates, we have $x_i^{k+1} = 0$ (meaning that none of the features in the ith block are used) if and only if

$$\left\| A_i^T (A_i x_i^k + \overline{z}^k - \overline{Ax}^k - u^k) \right\|_2 \leq \lambda/\rho.$$

When this occurs, the x_i-update is fast (compared to the case when $x_i^{k+1} \neq 0$). In a parallel implementation, there is no benefit to speeding up only some of the tasks being executed in parallel, but in a serial setting we do benefit.

8.3.2 Group Lasso

Consider the group lasso problem with the feature groups coinciding with the blocks of features, and ℓ_2 norm (*not* squared) regularization:

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda \textstyle\sum_{i=1}^N \|x_i\|_2.$$

The z-update and u-update are the same as for the lasso, but the x_i update becomes

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left((\rho/2)\|A_i x_i - A_i x_i^k - \overline{z}^k + \overline{Ax}^k + u^k\|_2^2 + \lambda\|x_i\|_2 \right).$$

(Only the subscript on the last norm differs from the lasso case.) This involves minimizing a function of the form

$$(\rho/2)\|A_i x_i - v\|_2^2 + \lambda\|x_i\|_2,$$

which can be carried out as follows. The solution is $x_i = 0$ if and only if $\|A_i^T v\|_2 \leq \lambda/\rho$. Otherwise, the solution has the form

$$x_i = (A_i^T A_i + \nu I)^{-1} A_i^T v,$$

for the value of $\nu > 0$ that gives $\nu\|x_i\|_2 = \lambda/\rho$. This value can be found using a one-parameter search (*e.g.*, via bisection) over ν. We can speed up the computation of x_i for several values of ν (as needed for the parameter search) by computing and caching an eigendecomposition of $A_i^T A_i$. Assuming A_i is tall, *i.e.*, $m \geq n_i$ (a similar method works when $m < n_i$), we compute an orthogonal Q for which $A_i^T A_i = Q\,\mathbf{diag}(\lambda)Q^T$, where λ is the vector of eigenvalues of $A_i^T A_i$ (*i.e.*, the squares of the singular values of A_i). The cost is $O(mn_i^2)$ flops, dominated (in order) by forming $A_i^T A_i$. We subsequently compute $\|x_i\|_2$ using

$$\|x_i\|_2 = \left\|\mathbf{diag}(\lambda + \nu\mathbf{1})^{-1} Q^T A_i^T v\right\|_2.$$

This can be computed in $O(n_i)$ flops, once $Q^T A_i^T v$ is computed, so the search over ν is costless (in order). The cost per iteration is thus $O(mn_i)$ (to compute $Q^T A_i^T v$), a factor of n_i better than carrying out the x_i-update without caching.

8.3.3 Sparse Logistic Regression

The algorithm is identical to the lasso problem above, except that the $\overline{z}$-update becomes

$$\overline{z}^{k+1} := \operatorname*{argmin}_{\overline{z}} \left(l(N\overline{z}) + (\rho/2)\|\overline{z} - \overline{Ax}^{k+1} - u^k\|_2^2 \right),$$

where l is the logistic loss function. This splits to the component level, and involves the proximity operator for l. This can be very efficiently computed by a lookup table that gives the approximate value, followed by one or two Newton steps (for a scalar problem).

It is interesting to see that in distributed sparse logistic regression, the dominant computation is the solution of N parallel *lasso* problems.

8.3.4 Support Vector Machine

The algorithm is

$$\begin{aligned}
x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left((\rho/2)\|A_i x_i - A_i x_i^k - \overline{z}^k + \overline{Ax}^k + u^k\|_2^2 + \lambda\|x_i\|_2^2 \right) \\
\overline{z}^{k+1} &:= \operatorname*{argmin}_{\overline{z}} \left(\mathbf{1}^T (N\overline{z} + \mathbf{1})_+ + (\rho/2)\|\overline{z} - \overline{Ax}^{k+1} - u^k\|_2^2 \right) \\
u^{k+1} &:= u^k + \overline{Ax}^{k+1} - \overline{z}^{k+1}.
\end{aligned}$$

The x_i-updates involve quadratic functions, and require solving ridge regression problems. The $\overline{z}$-update splits to the component level, and can be expressed as the shifted soft thresholding operation

$$\overline{z}_i^{k+1} := \begin{cases} v_i - N/\rho & v_i > -1/N + N/\rho \\ -1/N & v_i \in [-1/N, -1/N + N/\rho] \\ v_i & v_i < -1/N, \end{cases}$$

where $v = \overline{Ax}^{k+1} + \overline{u}^k$ (and here, the subscript denotes the entry in the vector $\overline{z}^{k+1}$).

8.3.5 Generalized Additive Models

A *generalized additive model* has the form

$$b \approx \sum_{j=1}^{n} f_j(a_j),$$

where a_j is the jth element of the feature vector a, and $f_j : \mathbf{R} \to \mathbf{R}$ are the feature functions. When the feature functions f_j are linear, *i.e.*, of the form $f_j(a_j) = w_j a_j$, this reduces to standard linear regression.

We choose the feature functions by solving the optimization problem

$$\text{minimize} \quad \sum_{i=1}^{m} l_i(\sum_{j=1}^{n} f_j(a_{ij}) - b_i) + \sum_{j=1}^{n} r_j(f_j),$$

where a_{ij} is the jth component of the feature vector of the ith example, and b_i is the associated outcome. Here the optimization variables are the functions $f_j \in \mathcal{F}_j$, where $\mathcal{F}_j$ is a subspace of functions; r_j is now a regularization functional. Usually f_j is linearly parametrized by a finite number of coefficients, which are the underlying optimization variables, but this formulation can also handle the case when $\mathcal{F}_j$ is infinite-dimensional. In either case, it is clearer to think of the feature functions f_j as the variables to be determined.

We split the features down to individual functions so $N = n$. The algorithm is

$$
\begin{aligned}
f_j^{k+1} :=& \\
&\underset{f_j \in \mathcal{F}_j}{\operatorname{argmin}} \left(r_j(f_j) + (\rho/2) \textstyle\sum_{i=1}^m (f_j(a_{ij}) - f_j^k(a_{ij}) - \overline{z}_i^k + \overline{f}_i^k + u_i^k)^2 \right) \\
\overline{z}^{k+1} :=& \underset{\overline{z}}{\operatorname{argmin}} \left(\textstyle\sum_{i=1}^m l_i(N\overline{z}_i - b_i) + (\rho/2) \sum_{i=1}^N \|\overline{z} - \overline{f}^{k+1} - u^k\|_2^2 \right) \\
u^{k+1} :=& \; u^k + \overline{f}^{k+1} - \overline{z}^{k+1},
\end{aligned}
$$

where $\overline{f}_i^k = (1/n) \sum_{j=1}^n f_j^k(a_{ij})$, the average value of the predicted response $\sum_{j=1}^n f_j^k(a_{ij})$ for the ith feature.

The f_j-update is an ℓ_2 (squared) regularized function fit. The $\overline{z}$-update can be carried out componentwise.

9

Nonconvex Problems

We now explore the use of ADMM for *nonconvex* problems, focusing on cases in which the individual steps in ADMM, *i.e.*, the x- and z-updates, can be carried out exactly. Even in this case, ADMM need not converge, and when it does converge, it need not converge to an optimal point; it must be considered just another local optimization method. The hope is that it will possibly have better convergence properties than other local optimization methods, where 'better convergence' can mean faster convergence or convergence to a point with better objective value. For nonconvex problems, ADMM can converge to different (and in particular, nonoptimal) points, depending on the initial values x^0 and y^0 and the parameter ρ.

9.1 Nonconvex Constraints

Consider the constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{S}, \end{array}$$

with f convex, but $\mathcal{S}$ nonconvex. Here, ADMM has the form

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - z^k + u^k\|_2^2 \right) \\ z^{k+1} &:= \Pi_{\mathcal{S}}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1}, \end{aligned}$$

where $\Pi_{\mathcal{S}}$ is projection onto $\mathcal{S}$. The x-minimization step (which is evaluating a proximal operator) is convex since f is convex, but the z-update is projection onto a nonconvex set. In general, this is hard to compute, but it can be carried out exactly in some important special cases we list below.

- *Cardinality.* If $\mathcal{S} = \{x \mid \mathbf{card}(x) \leq c\}$, where **card** gives the number of nonzero elements, then $\Pi_{\mathcal{S}}(v)$ keeps the c largest magnitude elements and zeroes out the rest.
- *Rank.* If $\mathcal{S}$ is the set of matrices with rank c, then $\Pi_{\mathcal{S}}(v)$ is determined by carrying out a singular value decomposition, $v = \sum_i \sigma_i u_i v_i^T$, and keeping the top c dyads, *i.e.*, form $\Pi_{\mathcal{S}}(v) = \sum_{i=1}^{c} \sigma_i u_i v_i^T$.
- *Boolean constraints.* If $\mathcal{S} = \{x \mid x_i \in \{0,1\}\}$, then $\Pi_{\mathcal{S}}(v)$ simply rounds each entry to 0 or 1, whichever is closer. Integer constraints can be handled in the same way.

9.1.1 Regressor Selection

As an example, consider the least squares *regressor selection* or *feature selection* problem,

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 \\ \text{subject to} & \mathbf{card}(x) \leq c, \end{array}$$

which is to find the best fit to b as a linear combination of no more than c columns of A. For this problem, ADMM takes the form above, where the x-update involves a regularized least squares problem, and the z-update involves keeping the c largest magnitude elements of $x^{k+1} + u^k$. This is just like ADMM for the lasso, except that soft thresholding is

replaced with 'hard' thresholding. This close connection is hardly surprising, since lasso can be thought of as a heuristic for solving the regressor selection problem. From this viewpoint, the lasso controls the trade-off between least squares error and sparsity through the parameter λ, whereas in ADMM for regressor selection, the same trade-off is controlled by the parameter c, the exact cardinality desired.

9.1.2 Factor Model Fitting

The goal is to approximate a symmetric matrix Σ (say, an empirical covariance matrix) as a sum of a rank-k and a diagonal positive semidefinite matrix. Using the Frobenius norm to measure approximation error, we have the problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|X + \mathbf{diag}(d) - \Sigma\|_F^2 \\ \text{subject to} & X \geq 0, \quad \mathbf{Rank}(X) = k, \quad d \geq 0, \end{array}$$

with variables $X \in \mathbf{S}^n$, $d \in \mathbf{R}^n$. (Any convex loss function could be used in lieu of the Frobenius norm.)

We take

$$\begin{aligned} f(X) &= \inf_{d \geq 0} (1/2)\|X + \mathbf{diag}(d) - \Sigma\|_F^2 \\ &= (1/2)\sum_{i \neq j} (X_{ij} - \Sigma_{ij})^2 + (1/2)\sum_{i=1}^n (X_{ii} - \Sigma_{ii})_+^2, \end{aligned}$$

with the optimizing d having the form $d_i = (\Sigma_{ii} - X_{ii})_+$, $i = 1, \ldots, n$. We take $\mathcal{S}$ to be the set of positive semidefinite rank-k matrices.

ADMM for the factor model fitting problem is then

$$\begin{aligned} X^{k+1} &:= \operatorname*{argmin}_X \left(f(X) + (\rho/2)\|X - Z^k + U^k\|_F^2 \right) \\ Z^{k+1} &:= \Pi_{\mathcal{S}}(X^{k+1} + U^k) \\ U^{k+1} &:= U^k + X^{k+1} - Z^{k+1}, \end{aligned}$$

where Z, $U \in \mathbf{S}^n$. The X-update is separable to the component level, and can be expressed as

$$\begin{array}{lcll} (X^{k+1})_{ij} & := & (1/(1+\rho)) \left(\Sigma_{ij} + \rho(Z_{ij}^k - U_{ij}^k) \right) & i \neq j \\ (X^{k+1})_{ii} & := & (1/(1+\rho)) \left(\Sigma_{ii} + \rho(Z_{ii}^k - U_{ii}^k) \right) & \Sigma_{ii} \leq Z_{ii} - U_{ii} \\ (X^{k+1})_{ii} & := & Z_{ii}^k - U_{ii}^k & \Sigma_{ii} > Z_{ii} - U_{ii}. \end{array}$$

The Z-update is carried out by an eigenvalue decomposition, keeping only the dyads associated with the largest k positive eigenvalues.

9.2 Bi-convex Problems

Another problem that admits exact ADMM updates is the general bi-convex problem,

$$\begin{array}{ll} \text{minimize} & F(x,z) \\ \text{subject to} & G(x,z) = 0, \end{array}$$

where $F : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ is *bi-convex*, *i.e.*, convex in x for each z and convex in z for each x, and $G : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^p$ is *bi-affine*, *i.e.*, affine in x for each fixed z, and affine in z for each fixed x. When F is separable in x and z, and G is jointly affine in x and z, this reduces to the standard ADMM problem form (3.1). For this problem ADMM has the form

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(F(x, z^k) + (\rho/2)\|G(x, z^k) + u^k\|_2^2 \right) \\ z^{k+1} &:= \operatorname*{argmin}_z \left(F(x^{k+1}, z) + (\rho/2)\|G(x^{k+1}, z) + u^k\|_2^2 \right) \\ u^{k+1} &:= u^k + G(x^{k+1}, z^{k+1}). \end{aligned}$$

Both the x- and z-updates involve convex optimization problems, and so are tractable.

When $G = 0$ (or is simply absent), ADMM reduces to simple alternating minimization, a standard method for bi-convex minimization.

9.2.1 Nonnegative Matrix Factorization

As an example, consider *nonnegative matrix factorization* [110]:

$$\begin{array}{ll} \text{minimize} & (1/2)\|VW - C\|_F^2 \\ \text{subject to} & V_{ij} \geq 0, \quad W_{ij} \geq 0, \end{array}$$

with variables $V \in \mathbf{R}^{p \times r}$ and $W \in \mathbf{R}^{r \times q}$, and data $C \in \mathbf{R}^{p \times q}$. In this form of the problem, the objective (which includes the constraints) is bi-affine, and there are no equality constraints, so ADMM becomes the standard method for nonnegative matrix factorization, which is

alternately minimizing over V, with W fixed, and then minimizing over W, with V fixed.

We can also introduce a new variable, moving the bi-linear term from the objective into the constraints:

$$\begin{array}{ll} \text{minimize} & (1/2)\|X - C\|_F^2 + I_+(V) + I_+(W) \\ \text{subject to} & X - VW = 0, \end{array}$$

with variables X, V, W, where I_+ is the indicator function for elementwise nonnegative matrices. With (X, V) serving the role of x, and W serving the role of z above, ADMM becomes

$$\begin{aligned} (X^{k+1}, V^{k+1}) &:= \underset{X,V \geq 0}{\operatorname{argmin}} \left(\|X - C\|_F^2 + (\rho/2)\|X - VW^k + U^k\|_F^2 \right) \\ W^{k+1} &:= \underset{W \geq 0}{\operatorname{argmin}} \|X^{k+1} - V^{k+1}W + U^k\|_F^2 \\ U^{k+1} &:= U^k + X^{k+1} - V^{k+1}W^{k+1}. \end{aligned}$$

The first step splits across the rows of X and V, so can be performed by solving a set of quadratic programs, in parallel, to find each row of X and V separately; the second splits in the columns of W, so can be performed by solving parallel quadratic programs to find each column.

10

Implementation

This section addresses the implementation of ADMM in a distributed computing environment. For simplicity, we focus on the global consensus problem with regularization,

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + g(z) \\ \text{subject to} & x_i - z = 0, \end{array}$$

where f_i is the ith objective function term and g is the global regularizer. Extensions to the more general consensus case are mostly straightforward. We first describe an abstract implementation and then show how this maps onto a variety of software frameworks.

10.1 Abstract Implementation

We refer to x_i and u_i as *local variables* stored in subsystem i, and to z as the *global variable*. For a distributed implementation, it is often more natural to group the local computations (*i.e.*, the x_i- and u_i-updates), so we write ADMM as

$$\begin{aligned} u_i &:= u_i + x_i - z \\ x_i &:= \operatorname{argmin}\left(f_i(x_i) + (\rho/2)\|x_i - z + u_i\|_2^2\right) \\ z &:= \mathbf{prox}_{g,N\rho}(\overline{x} + \overline{u})\,. \end{aligned}$$

Here, the iteration indices are omitted because in an actual implementation, we can simply overwrite previous values of these variables. Note that the u-update must be done before the x-update in order to match (7.6-7.8). If $g = 0$, then the z-update simply involves computing $\overline{x}$, and the u_i are not part of the aggregation, as discussed in §7.1.

This suggests that the main features required to implement ADMM are the following:

- *Mutable state.* Each subsystem i must store the current values of x_i and u_i.
- *Local computation.* Each subsystem must be able to solve a small convex problem, where 'small' means that the problem is solvable using a serial algorithm. In addition, each local process must have local access to whatever data are required to specify f_i.
- *Global aggregation.* There must be a mechanism for averaging local variables and broadcasting the result back to each subsystem, either by explicitly using a central collector or via some other approach like *distributed averaging* [160, 172]. If computing z involves a proximal step (*i.e.*, if g is nonzero), this can either be performed centrally or at each local node; the latter is easier to implement in some frameworks.
- *Synchronization.* All the local variables must be updated before performing global aggregation, and the local updates must all use the latest global variable. One way to implement this synchronization is via a *barrier*, a system checkpoint at which all subsystems must stop and wait until all other subsystems reach it.

When actually implementing ADMM, it helps to consider whether to take the 'local perspective' of a subsystem performing local processing and communicating with a central collector, or the 'global perspective' of a central collector coordinating the work of a set of subsystems. Which is more natural depends on the software framework used.

From the local perspective, each node i receives z, updates u_i and then x_i, sends them to the central collector, waits, and then receives the

updated z. From the global perspective, the central collector broadcasts z to the subsystems, waits for them to finish local processing, gathers all the x_i and u_i, and updates z. (Of course, if ρ varies across iterations, then ρ must also be updated and broadcast when z is updated.) The nodes must also evaluate the stopping criteria and decide when to terminate; see below for examples.

In the general form consensus case, which we do not discuss here, a decentralized implementation is possible that does not require z to be centrally stored; each set of subsystems that share a variable can communicate among themselves directly. In this setting, it can be convenient to think of ADMM as a message-passing algorithm on a graph, where each node corresponds to a subsystem and the edges correspond to shared variables.

10.2 MPI

Message Passing Interface (MPI) [77] is a language-independent message-passing specification used for parallel algorithms, and is the most widely used model for high-performance parallel computing today. There are numerous implementations of MPI on a variety of distributed platforms, and interfaces to MPI are available from a wide variety of languages, including C, C++, and Python.

There are multiple ways to implement consensus ADMM in MPI, but perhaps the simplest is given in Algorithm 1. This pseudocode uses a *single program, multiple data* (SPMD) programming style, in which each processor or subsystem runs the same program code but has its own set of local variables and can read in a separate subset of the data. We assume there are N processors, with each processor i storing local variables x_i and u_i, a (redundant) copy of the global variable z, and handling only the local data implicit in the objective component f_i.

In step 4, *Allreduce* denotes using the `MPI_Allreduce` operation to compute the global sum over all processors of the contents of the vector w, and store the result in w on every processor; the same applies to the scalar t. After step 4, then, $w = \sum_{i=1}^{n}(x_i + u_i) = N(\overline{x} + \overline{u})$ and $t = \|r\|_2^2 = \sum_{i=1}^{n} \|r_i\|_2^2$ on all processors. We use *Allreduce* because

Algorithm 1 Global consensus ADMM in MPI.

initialize N processes, along with x_i, u_i, r_i, z.

repeat

1. Update $u_i := u_i + x_i - z$.
2. Update $x_i := \text{argmin}_x \left(f_i(x) + (\rho/2)\|x - z + u_i\|_2^2\right)$.
3. Let $w := x_i + u_i$ and $t := \|r_i\|_2^2$.
4. *Allreduce* w and t.
5. Let $z^{\text{prev}} := z$ and update $z := \mathbf{prox}_{g,N\rho}(w/N)$.
6. **exit if** $\rho\sqrt{N}\|z - z^{\text{prev}}\|_2 \leq \epsilon^{\text{conv}}$ and $\sqrt{t} \leq \epsilon^{\text{feas}}$.
7. Update $r_i := x_i - z$.

its implementation is in general much more scalable than simply having each subsystem send its results directly to an explicit central collector.

Next, in steps 5 and 6, all processors (redundantly) compute the z-update and perform the termination test. It is possible to have the z-update and termination test performed on just one processor and broadcast the results to the other processors, but doing so complicates the code and is generally no faster.

10.3 Graph Computing Frameworks

Since ADMM can be interpreted as performing message-passing on a graph, it is natural to implement it in a graph processing framework. Conceptually, the implementation will be similar to the MPI case discussed above, except that the role of the central collector will often be handled abstractly by the system, rather than having an explicit central collector process. In addition, higher-level graph processing frameworks provide a number of built in services that one would otherwise have to manually implement, such as fault tolerance.

Many modern graph frameworks are based on or inspired by Valiant's *bulk-synchronous parallel* (BSP) model [164] for parallel computation. A BSP computer consists of a set of processors networked together, and a BSP computation consists of a series of global *supersteps*. Each superstep consists of three stages: *parallel*

computation, in which the processors, in parallel, perform local computations; *communication*, in which the processors communicate among themselves; and *barrier synchronization*, in which the processes wait until all processes are finished communicating.

The first step in each ADMM superstep consists of performing local u_i- and x_i-updates. The communication step would broadcast the new x_i and u_i values to a central collector node, or globally to each individual processor. Barrier synchronization is then used to ensure that all the processors have updated their primal variable before the central collector averages and rebroadcasts the results.

Specific frameworks directly based on or inspired by the BSP model include the Parallel BGL [91], GraphLab [114], and Pregel [119], among others. Since all three follow the general outline above, we refer the reader to the individual papers for details.

10.4 MapReduce

MapReduce [46] is a popular programming model for distributed batch processing of very large datasets. It has been widely used in industry and academia, and its adoption has been bolstered by the open source project Hadoop, inexpensive cloud computing services available through Amazon, and enterprise products and services offered by Cloudera. MapReduce libraries are available in many languages, including Java, C++, and Python, among many others, though Java is the primary language for Hadoop. Though it is awkward to express ADMM in MapReduce, the amount of cloud infrastructure available for MapReduce computing can make it convenient to use in practice, especially for large problems. We briefly review some key features of Hadoop below; see [170] for general background.

A MapReduce computation consists of a set of Map tasks, which process subsets of the input data in parallel, followed by a Reduce task, which combines the results of the Map tasks. Both the Map and Reduce functions are specified by the user and operate on key-value pairs. The Map function performs the transformation

$$(k, v) \mapsto [(k'_1, v'_1), \ldots, (k'_m, v'_m)],$$

that is, it takes a key-value pair and emits a list of intermediate key-value pairs. The engine then collects all the values $v'_1, \ldots, v'_r$ that correspond to the same output key k' (across all Mappers) and passes them to the Reduce functions, which performs the transformation

$$(k', [v'_1, \ldots, v'_r]) \mapsto (k'', R(v'_1, \ldots, v'_r)),$$

where R is a commutative and associative function. For example, R could simply sum v'_i. In Hadoop, Reducers can emit lists of key-value pairs rather than just a single pair.

Each iteration of ADMM can easily be represented as a MapReduce task: The parallel local computations are performed by Maps, and the global aggregation is performed by a Reduce. We will describe a simple global consensus implementation to give the general flavor and discuss the details below. Here, we have the Reducer compute

Algorithm 2 An iteration of global consensus ADMM in Hadoop/ MapReduce.

function map(key i, dataset $\mathcal{D}_i$)
1. Read $(x_i, u_i, \hat{z})$ from HBase table.
2. Compute $z := \mathbf{prox}_{g, N\rho}((1/N)\hat{z})$.
3. Update $u_i := u_i + x_i - z$.
4. Update $x_i := \operatorname{argmin}_x \left(f_i(x) + (\rho/2)\|x - z + u_i\|_2^2\right)$.
5. *Emit* (key CENTRAL, record (x_i, u_i)).

function reduce(key CENTRAL, records $(x_1, u_1), \ldots, (x_N, u_N)$)
1. Update $\hat{z} := \sum_{i=1}^N x_i + u_i$.
2. *Emit* (key j, record $(x_j, u_j, \hat{z})$) to HBase for $j = 1, \ldots, N$.

$\hat{z} = \sum_{i=1}^N (x_i + u_i)$ rather than z or $\tilde{z}$ because summation is associative while averaging is not. We assume N is known (or, alternatively, the Reducer can compute the sum $\sum_{i=1}^N 1$). We have N Mappers, one for each subsystem, and each Mapper updates u_i and x_i using the $\hat{z}$ from the previous iteration. Each Mapper independently executes the proximal step to compute z, but this is usually a cheap operation like soft thresholding. It emits an intermediate key-value pair that essentially serves as a message to the central collector. There is a single Reducer, playing the role of a central collector, and its incoming values are the messages from the Mappers. The updated records are

then written out directly to HBase by the Reducer, and a wrapper program restarts a new MapReduce iteration if the algorithm has not converged. The wrapper will check whether $\rho\sqrt{N}\|z - z^{\text{prev}}\|_2 \leq \epsilon^{\text{conv}}$ and $(\sum_{i=1}^{N} \|x_i - z\|_2^2)^{1/2} \leq \epsilon^{\text{feas}}$ to determine convergence, as in the MPI case. (The wrapper checks the termination criteria instead of the Reducer because they are not associative to check.)

The main difficulty is that MapReduce tasks are not designed to be iterative and do not preserve state in the Mappers across iterations, so implementing an iterative algorithm like ADMM requires some understanding of the underlying infrastructure. Hadoop contains a number of components supporting large-scale, fault-tolerant distributed computing applications. The relevant components here are HDFS, a distributed file system based on Google's GFS [85], and HBase, a distributed database based on Google's BigTable [32].

HDFS is a distributed filesystem, meaning that it manages the storage of data across an entire cluster of machines. It is designed for situations where a typical file may be gigabytes or terabytes in size and high-speed streaming read access is required. The base units of storage in HDFS are *blocks*, which are 64 MB to 128 MB in size in a typical configuration. Files stored on HDFS are comprised of blocks; each block is stored on a particular machine (though for redundancy, there are replicas of each block on multiple machines), but different blocks in the same file need not be stored on the same machine or even nearby. For this reason, any task that processes data stored on HDFS (*e.g.*, the local datasets $\mathcal{D}_i$) should process a single block of data at a time, since a block is guaranteed to reside wholly on one machine; otherwise, one may cause unnecessary network transfer of data.

In general, the input to each Map task is data stored on HDFS, and Mappers cannot access local disk directly or perform any stateful computation. The scheduler runs each Mapper as close to its input data as possible, ideally on the same node, in order to minimize network transfer of data. To help preserve data locality, each Map task should also be assigned around a block's worth of data. Note that this is very different from the implementation presented for MPI, where each process can be told to pick up the local data on whatever machine it is running on.

Since each Mapper only handles a single block of data, there will usually be a number of Mappers running on the same machine. To reduce the amount of data transferred over the network, Hadoop supports the use of *combiners*, which essentially Reduce the results of all the Map tasks on a given node so only one set of intermediate key-value pairs need to be transferred across machines for the final Reduce task. In other words, the Reduce step should be viewed as a two-step process: First, the results of all the Mappers on each individual node are reduced with Combiners, and then the records across each machine are Reduced. This is a major reason why the Reduce function must be commutative and associative.

Since the input value to a Mapper is a block of data, we also need a mechanism for a Mapper to read in local variables, and for the Reducer to store the updated variables for the next iteration. Here, we use HBase, a distributed database built on top of HDFS that provides fast random read-write access. HBase, like BigTable, provides a distributed multi-dimensional sorted map. The map is indexed by a row key, a column key, and a timestamp. Each cell in an HBase table can contain multiple versions of the same data indexed by timestamp; in our case, we can use the iteration counts as the timestamps to store and access data from previous iterations; this is useful for checking termination criteria, for example. The row keys in a table are strings, and HBase maintains data in lexicographic order by row key. This means that rows with lexicographically adjacent keys will be stored on the same machine or nearby. In our case, variables should be stored with the subsystem identifier at the beginning of row key, so information for the same subsystem is stored together and is efficient to access. For more details, see [32, 170].

The discussion and pseudocode above omits and glosses over many details for simplicity of exposition. MapReduce frameworks like Hadoop also support much more sophisticated implementations, which may be necessary for very large scale problems. For example, if there are too many values for a single Reducer to handle, we can use an approach analogous to the one suggested for MPI: Mappers emit pairs to 'regional' reduce jobs, and then an additional MapReduce step is carried out that uses an identity mapper and aggregates regional results

into a global result. In this section, our goal is merely to give a general flavor of some of the issues involved in implementing ADMM in a MapReduce framework, and we refer to [46, 170, 111] for further details. There has also been some recent work on alternative MapReduce systems that are specifically designed for iterative computation, which are likely better suited for ADMM [25, 179], though the implementations are less mature and less widely available. See [37, 93] for examples of recent papers discussing machine learning and optimization in MapReduce frameworks.

11

Numerical Examples

In this section we report numerical results for several examples. The examples are chosen to illustrate a variety of the ideas discussed above, including caching matrix factorizations, using iterative solvers for the updates, and using consensus and sharing ADMM to solve distributed problems. The implementations of ADMM are written to be as simple as possible, with no implementation-level optimization or tuning.

The first section discusses a small instance of the lasso problem with a dense coefficient matrix. This helps illustrate some of the basic behavior of the algorithm, and the impact of some of the linear algebra-based optimizations suggested in §4. We find, for example, that we can compute the entire regularization path for the lasso in not much more time than it takes to solve a single problem instance, which in turn takes not much more time than solving a single ridge regression problem of the same size.

We then discuss a serial implementation of the consensus ADMM algorithm applied to ℓ_1 regularized logistic regression, where we split the problem across training examples. Here, we focus on details of implementing consensus ADMM for this problem, rather than on actual

distributed solutions. The following section has a similar discussion of the group lasso problem, but split across features, with each regularization group corresponding to a distinct subsystem.

We then turn to a real large-scale distributed implementation using an MPI-based solver written in C. We report on the results of solving some large lasso problems on clusters hosted in Amazon EC2, and find that a fairly basic implementation is able to solve a lasso problem with 30 GB of data in a few minutes.

Our last example is regressor selection, a nonconvex problem. We compare the sparsity-fit trade-off curve obtained using nonconvex ADMM, directly controlling the number of regressors, with the same curve obtained using the lasso regularization path (with posterior least squares fit). We will see that the curves are not the same, but give very similar results. This suggests that the regressor selection method may be preferable to the lasso when the desired sparsity level is known in advance: It is much easier to explicitly set the desired sparsity level than tuning the regularization parameter λ to obtain this level.

All examples except the large-scale lasso are implemented in Matlab, and run on an Intel Core i3 processor running at 3.2 GHz. The large lasso example is implemented in C using MPI for interprocess communication and the GNU Scientific Library for linear algebra. Source code and data for these examples (and others) can be found at `www.stanford.edu/~boyd/papers/admm_distr_stats.html` and most are extensively commented.

11.1 Small Dense Lasso

We consider a small, dense instance of the lasso problem (6.2), where the feature matrix A has $m = 1500$ examples and $n = 5000$ features.

We generate the data as follows. We first choose $A_{ij} \sim \mathcal{N}(0,1)$ and then normalize the columns to have unit ℓ_2 norm. A 'true' value $x^{\text{true}} \in \mathbf{R}^n$ is generated with 100 nonzero entries, each sampled from an $\mathcal{N}(0,1)$ distribution. The labels b are then computed as $b = Ax^{\text{true}} + v$, where $v \sim \mathcal{N}(0, 10^{-3}I)$, which corresponds to a signal-to-noise ratio $\|Ax^{\text{true}}\|_2^2/\|v\|_2^2$ of around 60.

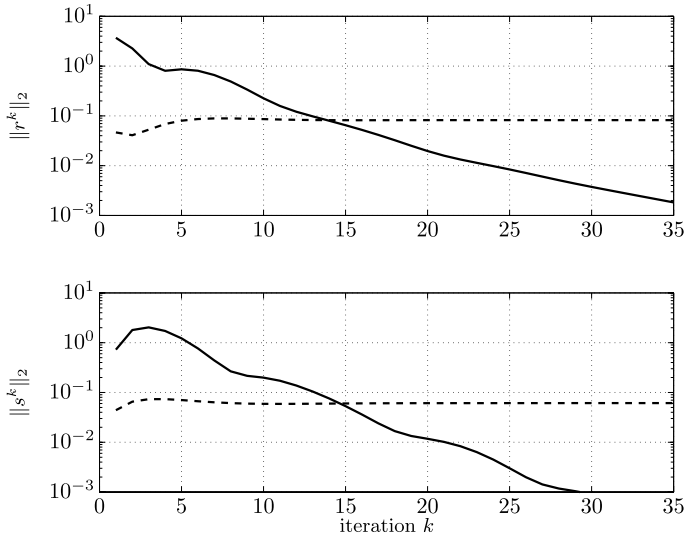

Fig. 11.1. Norms of primal residual (top) and dual residual (bottom) versus iteration, for a lasso problem. The dashed lines show ϵ^{pri} (top) and ϵ^{dual} (bottom).

We set the penalty parameter $\rho = 1$ and set termination tolerances $\epsilon^{\text{abs}} = 10^{-4}$ and $\epsilon^{\text{rel}} = 10^{-2}$. The variables u^0 and z^0 were initialized to be zero.

11.1.1 Single Problem

We first solve the lasso problem with regularization parameter $\lambda = 0.1\lambda_{\max}$, where $\lambda_{\max} = \|A^T b\|_\infty$ is the critical value of λ above which the solution of the lasso problem is $x = 0$. (Although not relevant, this choice correctly identifies about 80% of the nonzero entries in x^{true}.)

Figure 11.1 shows the primal and dual residual norms by iteration, as well as the associated stopping criterion limits ϵ^{pri} and ϵ^{dual} (which vary slightly in each iteration since they depend on x^k, z^k, and y^k through the relative tolerance terms). The stopping criterion was satisfied after 15 iterations, but we ran ADMM for 35 iterations to show the continued progress. Figure 11.2 shows the objective suboptimality $\tilde{p}^k - p^\star$, where $\tilde{p}^k = (1/2)\|Az^k - b\|_2^2 + \lambda\|z^k\|_1$ is the objective value at z^k. The optimal objective value $p^\star = 17.4547$ was independently verified using `l1_ls` [102].

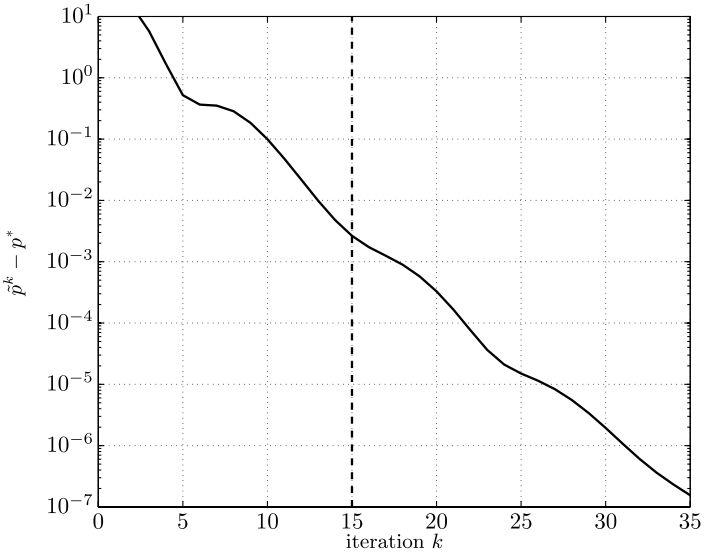

Fig. 11.2. Objective suboptimality versus iteration for a lasso problem. The stopping criterion is satisfied at iteration 15, indicated by the vertical dashed line.

Since A is fat (*i.e.*, $m < n$), we apply the matrix inversion lemma to $(A^TA + \rho I)^{-1}$ and instead compute the factorization of the smaller matrix $I + (1/\rho)AA^T$, which is then cached for subsequent x-updates. The factor step itself takes about $nm^2 + (1/3)m^3$ flops, which is the cost of forming AA^T and computing the Cholesky factorization. Subsequent updates require two matrix-vector multiplications and forward-backward solves, which require approximately $4mn + 2m^2$ flops. (The cost of the soft thresholding step in the z-update is negligible.) For these problem dimensions, the flop count analysis suggests a factor/solve ratio of around 350, which means that 350 subsequent ADMM iterations can be carried out for the cost of the initial factorization.

In our basic implementation, the factorization step takes about 1 second, and subsequent x-updates take around 30 ms. (This gives a factor/solve ratio of only 33, less than predicted, due to a particularly efficient matrix-matrix multiplication routine used in Matlab.) Thus the total cost of solving an entire lasso problem is around 1.5 seconds—only 50% more than the initial factorization. In terms of parameter estimation, we can say that computing the lasso estimate requires only 50%

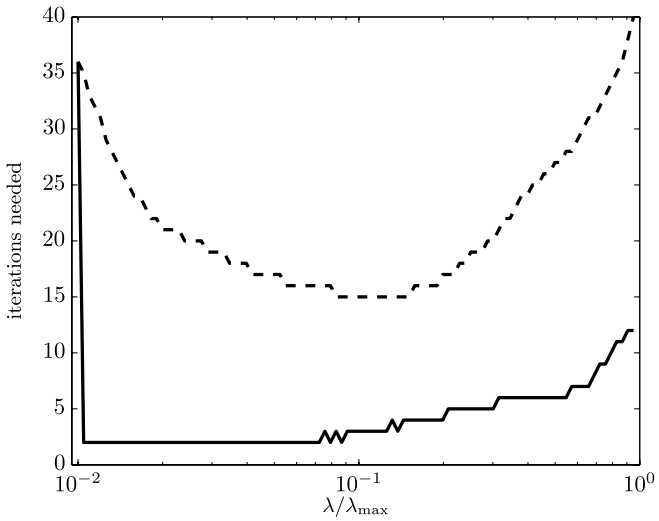

Fig. 11.3. Iterations needed versus λ for warm start (solid line) and cold start (dashed line).

more time than a ridge regression estimate. (Moreover, in an implementation with a higher factor/solve ratio, the additional effort for the lasso would have been even smaller.)

Finally, we report the effect of varying the parameter ρ on convergence time. Varying ρ over the 100:1 range from 0.1 to 10 yields a solve time ranging between 1.45 seconds to around 4 seconds. (In an implementation with a larger factor/solve ratio, the effect of varying ρ would have been even smaller.) Over-relaxation with $\alpha = 1.5$ does not significantly change the convergence time with $\rho = 1$, but it does reduce the worst convergence time over the range $\rho \in [0.1, 10]$ to only 2.8 seconds.

11.1.2 Regularization Path

To illustrate computing the regularization path, we solve the lasso problem for 100 values of λ, spaced logarithmically from $0.01\lambda_{\max}$ (where $x^\star$ has around 800 nonzeros) to $0.95\lambda_{\max}$ (where $x^\star$ has two nonzero entries). We first solve the lasso problem as above for $\lambda = 0.01\lambda_{\max}$, and for each subsequent value of λ, we then initialize (warm start) z and u at their optimal values for the previous λ. This requires only one factorization for all the computations; warm starting ADMM at the

Task	Time (s)
Factorization	1.1
x-Update	0.03
Single lasso ($\lambda = 0.1\lambda_{\max}$)	1.5
Cold start regularization path (100 values of λ)	160
Warm start regularization path (100 values of λ)	13

Table 11.1. Summary of timings for lasso example.

previous value significantly reduces the number of ADMM iterations required to solve each lasso problem after the first one.

Figure 11.3 shows the number of iterations required to solve each lasso problem using this warm start initialization, compared to the number of iterations required using a cold start of $z^0 = u^0 = 0$ for each λ. For the 100 values of λ, the total number of ADMM iterations required is 428, which takes 13 seconds in all. By contrast, with cold starts, we need 2166 total ADMM iterations and 100 factorizations to compute the regularization path, or around 160 seconds total. This timing information is summarized in Table 11.1.

11.2 Distributed ℓ_1 Regularized Logistic Regression

In this example, we use consensus ADMM to fit an ℓ_1 regularized logistic regression model. Following §8, the problem is

$$\text{minimize} \quad \sum_{i=1}^{m} \log\left(1 + \exp(-b_i(a_i^T w + v))\right) + \lambda \|w\|_1, \tag{11.1}$$

with optimization variables $w \in \mathbf{R}^n$ and $v \in \mathbf{R}$. The training set consists of m pairs (a_i, b_i), where $a_i \in \mathbf{R}^n$ is a feature vector and $b_i \in \{-1, 1\}$ is the corresponding label.

We generated a problem instance with $m = 10^6$ training examples and $n = 10^4$ features. The m examples are distributed among $N = 100$ subsystems, so each subsystem has 10^4 training examples. Each feature vector a_i was generated to have approximately 10 nonzero features, each sampled independently from a standard normal distribution. We chose a 'true' weight vector $w^{\text{true}} \in \mathbf{R}^n$ to have 100 nonzero values, and these entries, along with the true intercept v^{true}, were sampled

independently from a standard normal distribution. The labels b_i were then generated using

$$b_i = \mathbf{sign}(a_i^T w^{\text{true}} + v^{\text{true}} + v_i),$$

where $v_i \sim \mathcal{N}(0, 0.1)$.

The regularization parameter is set to $\lambda = 0.1\lambda_{\max}$, where $\lambda_{\max}$ is the critical value above which the solution of the problem is $w^\star = 0$. Here $\lambda_{\max}$ is more complicated to describe than in the simple lasso case described above. Let θ^{neg} be the fraction of examples with $b_i = -1$ and θ^{pos} the fraction with $b_i = 1$, and let $\tilde{b} \in \mathbf{R}^m$ be a vector with entries θ^{neg} where $b_i = 1$ and $-\theta^{\text{pos}}$ where $b_i = -1$. Then $\lambda_{\max} = \|A^T\tilde{b}\|_\infty$ (see [103, §2.1]). (While not relevant here, the final fitted model with $\lambda = 0.1\lambda_{\max}$ classified the training examples with around 90% accuracy.)

Fitting the model involves solving the global consensus problem (8.3) in §8.2 with local variables $x_i = (v_i, w_i)$ and consensus variable $z = (v, w)$. As in the lasso example, we used $\epsilon^{\text{abs}} = 10^{-4}$ and $\epsilon^{\text{rel}} = 10^{-2}$ as tolerances and used the initialization $u_i^0 = 0, z^0 = 0$. We use the penalty parameter value $\rho = 1$ for the iterations.

We used L-BFGS to carry out the x_i-updates. We used Nocedal's Fortran 77 implementation of L-BFGS with no tuning: We used default parameters, a memory of 5, and a constant termination tolerance across ADMM iterations (for a more efficient implementation, these tolerances would start large and decrease with ADMM iterations). We warm started the x_i-updates.

We used a serial implementation that performs the x_i-updates sequentially; in a distributed implementation, of course, the x_i-updates would be performed in parallel. To report an approximation of the timing that would have been achieved in a parallel implementation, we report the *maximum* time required to update x_i among the K subsystems. This corresponds roughly to the maximum number of L-BFGS iterations required for the x_i-update.

Figure 11.4 shows the progress of the primal and dual residual norm by iteration. The dashed line shows when the stopping criterion has been satisfied (after 19 iterations), resulting in a primal residual norm of about 1. Since the RMS consensus error can be expressed as $(1/\sqrt{m})\|r^k\|_2$ where $m = 10^6$, a primal residual norm of about 1 means that on average, the elements of x_i agree with z up to the third digit.

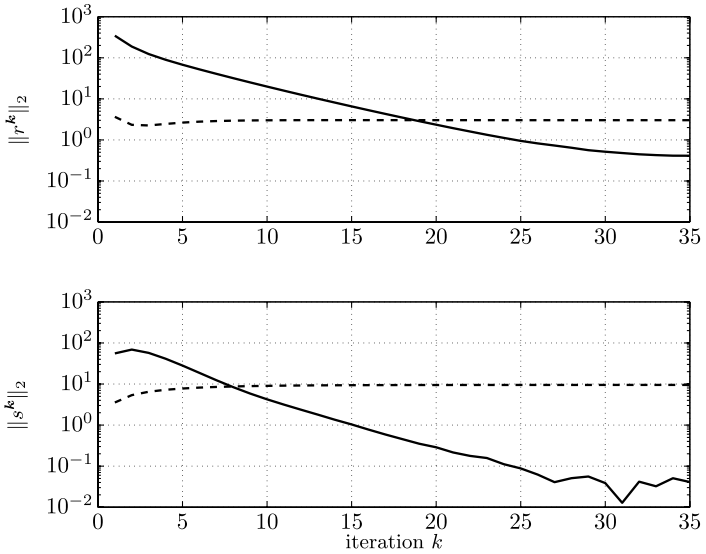

Fig. 11.4. Progress of primal and dual residual norm for distributed ℓ_1 regularized logistic regression problem. The dashed lines show ϵ^{pri} (top) and ϵ^{dual} (bottom).

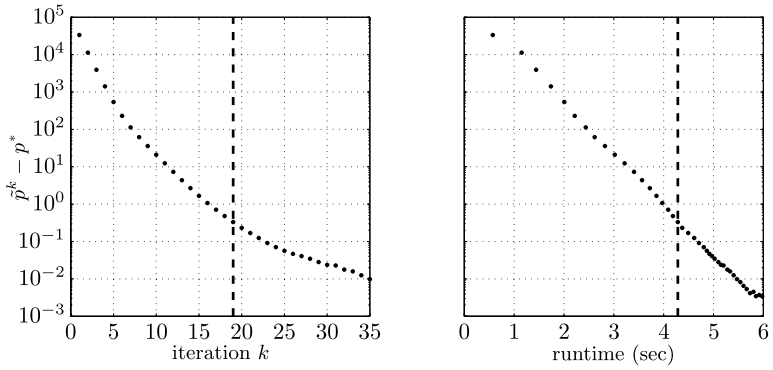

Fig. 11.5. *Left.* Objective suboptimality of distributed ℓ_1 regularized logistic regression versus iteration. *Right.* Progress versus elapsed time. The stopping criterion is satisfied at iteration 19, indicated by the vertical dashed line.

Figure 11.5 shows the suboptimality $\tilde{p}^k - p^\star$ for the consensus variable, where

$$\tilde{p}^k = \sum_{i=1}^{m} \log\left(1 + \exp(-b_i(a_i^T w^k + v^k))\right) + \lambda \|w^k\|_1.$$

The optimal value $p^\star = 0.3302 \times 10^6$ was verified using `l1_logreg` [103]. The lefthand plot shows ADMM progress by iteration, while the righthand plot shows the cumulative time in a parallel implementation. It took 19 iterations to satisfy the stopping criterion. The first 4 iterations of ADMM took 2 seconds, while the last 4 iterations (before the stopping criterion is satisfied) took less than 0.5 seconds. This is because as the iterates approach consensus, L-BFGS requires fewer iterations due to warm starting.

11.3 Group Lasso with Feature Splitting

We consider the group lasso example, described in §6.4.2,

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda \textstyle\sum_{i=1}^N \|x_i\|_2,$$

where $x = (x_1, \ldots, x_N)$, with $x_i \in \mathbf{R}^{n_i}$. We will solve the problem by splitting across feature groups $x_1, \ldots, x_N$ using the formulation in §8.3.

We generated a problem instance with $N = 200$ groups of features, with $n_i = 100$ features per group, for $i = 1, \ldots, 200$, for a total of $n = 20000$ features and $m = 200$ examples. A 'true' value $x^{\text{true}} \in \mathbf{R}^n$ was generated, with 9 nonzero groups, resulting in 900 nonzero feature values. The feature matrix A is dense, with entries drawn from an $\mathcal{N}(0,1)$ distribution, and its columns then normalized to have unit ℓ_2 norm (as in the lasso example of §11.1). The outcomes b are generated by $b = Ax^{\text{true}} + v$, where $v \sim \mathcal{N}(0, 0.1I)$, which corresponds to a signal-to-noise ratio $\|Ax^{\text{true}}\|_2^2/\|v\|_2^2$ of around 60.

We used the penalty parameter $\rho = 10$ and set termination tolerances as $\epsilon^{\text{abs}} = 10^{-4}$ and $\epsilon^{\text{rel}} = 10^{-2}$. The variables u^0 and z^0 are initialized to be zero.

We used the regularization parameter value $\lambda = 0.5\lambda_{\max}$, where

$$\lambda_{\max} = \max\{\|A_1^T b\|_2, \ldots, \|A_N^T b\|_2\}$$

is the critical value of λ above which the solution is $x = 0$. (Although not relevant, this choice of λ correctly identifies 6 of the 9 nonzero groups in x^{true} and produces an estimate with 17 nonzero groups.) The stopping criterion was satisfied after 47 iterations.

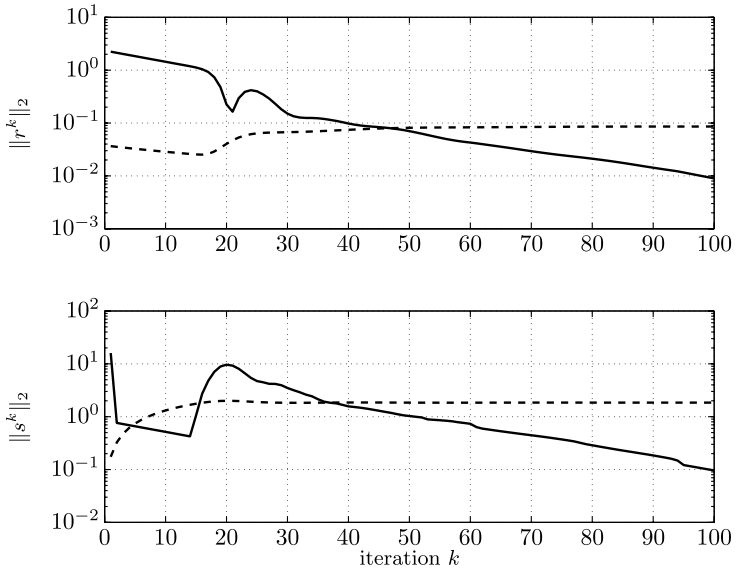

Fig. 11.6. Norms of primal residual (top) and dual residual (bottom) versus iteration, for the distributed group lasso problem. The dashed lines show ϵ^{pri} (top) and ϵ^{dual} (bottom).

The x_i-update is computed using the method described in §8.3.2, which involves computing and caching eigendecompositions of $A_i^T A_i$. The eigenvalue decompositions of $A_i^T A_i$ took around 7 milliseconds; subsequent x_i-updates took around 350 microseconds, around a factor of 20 faster. For 47 ADMM iterations, these numbers predict a total runtime in a serial implementation of about 5 seconds; the actual runtime was around 7 seconds. For a parallel implementation, we can estimate the runtime (neglecting interprocess communication and data distribution) as being about 200 times faster, around 35 milliseconds.

Figure 11.6 shows the progress of the primal and dual residual norm by iteration. The dashed line shows when the stopping criterion is satisfied (after 47 iterations).

Figure 11.7 shows the suboptimality $\tilde{p}^k - p^\star$ for the problem versus iteration, where

$$\tilde{p}^k = (1/2)\|Ax^k - b\|_2^2 + \lambda \sum_{i=1}^{K} \|x_i^k\|_2.$$

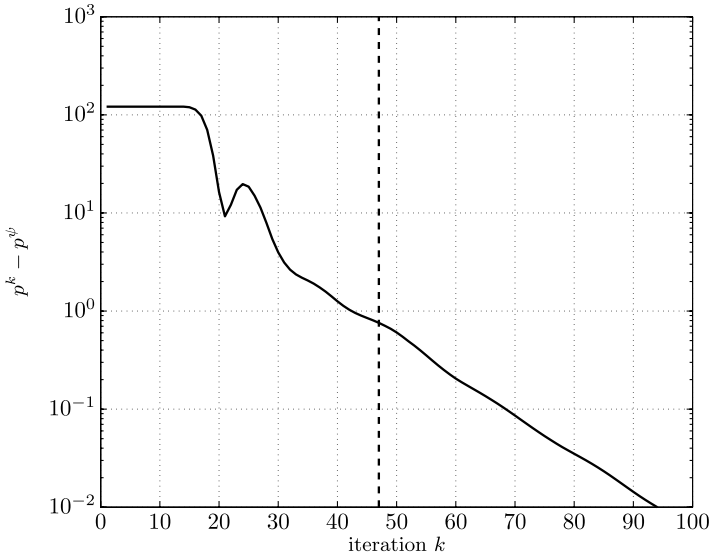

Fig. 11.7. Suboptimality of distributed group lasso versus iteration. The stopping criterion is satisfied at iteration 47, indicated by the vertical dashed line.

The optimal objective value $p^\star = 430.8390$ was found by running ADMM for 1000 iterations.

11.4 Distributed Large-Scale Lasso with MPI

In previous sections, we discussed an idealized version of a distributed implementation that was actually carried out serially for simplicity. We now turn to a much more realistic distributed example, in which we solve a very large instance of the lasso problem (6.2) using a distributed solver implemented in C using MPI for inter-process communication and the GNU Scientific Library (GSL) for linear algebra. In this example, we split the problem across training examples rather than features. We carried out the experiments in a cluster of virtual machines running on Amazon's Elastic Compute Cloud (EC2). Here, we focus entirely on scaling and implementation details.

The data was generated as in §11.1, except that we now solve a problem with $m = 400000$ examples and $n = 8000$ features across $N = 80$ subsystems, so each subsystem handles 5000 training examples. Note

that the overall problem has a skinny coefficient matrix but each of the subproblems has a fat coefficient matrix. We emphasize that the coefficient matrix is *dense*, so the full dataset requires over 30 GB to store and has 3.2 billion nonzero entries in the total coefficient matrix A. This is far too large to be solved efficiently, or at all, using standard serial methods on commonly available hardware.

We solved the problem using a cluster of 10 machines. We used Cluster Compute instances, which have 23 GB of RAM, two quad-core Intel Xeon X5570 'Nehalem' chips, and are connected to each other with 10 Gigabit Ethernet. We used hardware virtual machine images running CentOS 5.4. Since each node had 8 cores, we ran the code with 80 processes, so each subsystem ran on its own core. In MPI, communication between processes on the same machine is performed locally via the shared-memory Byte Transfer Layer (BTL), which provides low latency and high bandwidth communication, while communication across machines goes over the network. The data was sized so all the processes on a single machine could work entirely in RAM. Each node had its own attached Elastic Block Storage (EBS) volume that contained only the local data relevant to that machine, so disk throughput was shared among processes on the same machine but not across machines. This is to emulate a scenario where each machine is only processing the data on its local disk, and none of the dataset is transferred over the network. We emphasize that usage of a cluster set up in this fashion costs under $20 per hour.

We solved the problem with a deliberately naive implementation of the algorithm, based directly on the discussion of §6.4, §8.2, and §10.2. The implementation consists of a single file of C code, under 400 lines despite extensive comments. The linear algebra (BLAS operations and the Cholesky factorization) were performed using a stock installation of the GNU Scientific Library.

We now report the breakdown of the wall-clock runtime. It took roughly 30 seconds to load all the data into memory. It then took 4-5 minutes to form and then compute the Cholesky factorizations of $I + (1/\rho)A_i A_i^T$. After caching these factorizations, it then took 0.5-2 seconds for each subsequent ADMM iteration. This includes the back-solves in the x_i-updates and all the message passing. For this problem,

Total dataset size	30 GB
Number of subsystems	80
Total dataset dimensions	400000×8000
Subsystem dimensions	5000×8000
Data loading time	30 seconds
Factorization time	5 minutes
Single iteration time	1 second
Total runtime	6 minutes

Table 11.2. Rough summary of a large dense distributed lasso example.

ADMM converged in 13 iterations, yielding a start-to-finish runtime of under 6 minutes to solve the whole problem. Approximate times are summarized in Table 11.2.

Though we did not compute it as part of this example, the extremely low cost of each iteration means that it would be straightforward to compute the entire regularization path for this problem using the method described in §11.1.2. In that example, it required 428 iterations to compute the regularization path for 100 settings of λ, while it took around 15 iterations for a single instance to converge, roughly the same as in this example. Extrapolating for this case, it is plausible that the entire regularization path, even for this very large problem, could easily be obtained in another five to ten minutes.

It is clear that by far the dominant computation is forming and computing the Cholesky factorization, locally and in parallel, of each $A_i^T A_i + \rho I$ (or $I + (1/\rho) A_i A_i^T$, if the matrix inversion lemma is applied). As a result, it is worth keeping in mind that the performance of the linear algebra operations in our basic implementation can be significantly improved by using LAPACK instead of GSL for the Cholesky factorization, and by replacing GSL's BLAS implementation with a hardware-optimized BLAS library produced by ATLAS, a vendor library like Intel MKL, or a GPU-based linear algebra package. This could easily lead to several orders of magnitude faster performance.

In this example, we used a dense coefficient matrix so the code could be written using a single simple math library. Many real-world examples of the lasso have larger numbers of training examples or features, but are sparse and do not have billions of nonzero entries, as we do here. The code we provide could be modified in the usual manner

to handle sparse or structured matrices (*e.g.*, use CHOLMOD [35] for sparse Cholesky factorization), and would also scale to very large problems. More broadly, it could also be adapted with minimal work to add constraints or otherwise modify the lasso problem, or even solve completely different problems, like training logistic regression or SVMs.

It is worth observing that ADMM scales well both horizontally and vertically. We could easily have solved much larger problem instances in roughly the same amount of time than the one described here by having each subsystem solve a larger subproblem (up to the point where each machine's RAM is saturated, which it was not here); by running more subsystems on each machine (though this can lead to performance degradation in key areas like the factorization step); or simply by adding more machines to the cluster, which is mostly straightforward and relatively inexpensive on Amazon EC2.

Up to a certain problem size, the solver can be implemented by users who are not expert in distributed systems, distributed linear algebra, or advanced implementation-level performance enhancements. This is in sharp contrast to what is required in many other cases. Solving extremely large problem instances requiring hundreds or thousands of machines would require a more sophisticated implementation from a systems perspective, but it is interesting to observe that a basic version can solve rather large problems quickly on standard software and hardware. To the best of our knowledge, the example above is one of the largest lasso problems ever solved.

11.5 Regressor Selection

In our last example, we apply ADMM to an instance of the (nonconvex) least squares regressor selection problem described in §9.1, which seeks the best quadratic fit to a set of labels b from a combination of no more than c columns of A (regressors). We use the same A and b generated for the dense lasso example in §11.1, with $m = 1500$ examples and $n = 5000$ features, but instead of relying on the ℓ_1 regularization heuristic to achieve a sparse solution, we explicitly constrain its cardinality to be below $c = 100$.

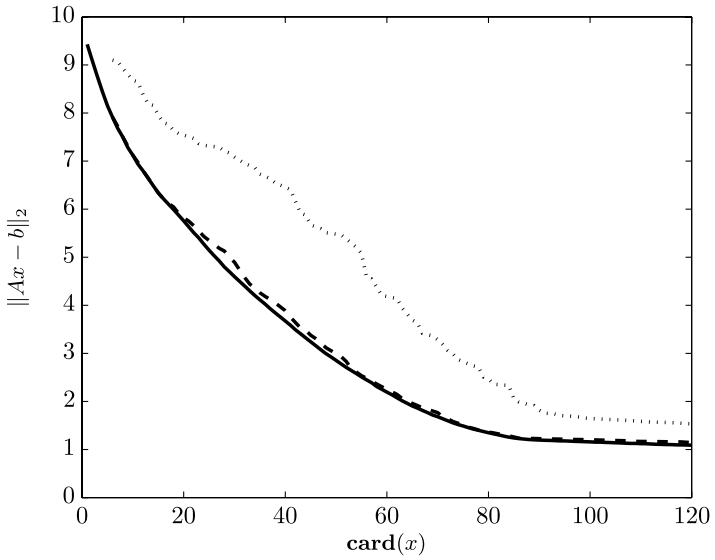

Fig. 11.8. Fit versus cardinality for the lasso (dotted line), lasso with posterior least squares fit (dashed line), and regressor selection (solid line).

The x-update step has exactly the same expression as in the lasso example, so we use the same method, based on the matrix inversion lemma and caching, described in that example. The z-update step consists of keeping the c largest magnitude components of $x + u$ and zeroing the rest. For the sake of clarity, we performed an intermediate sorting of the components, but more efficient schemes are possible. In any case, the cost of the z-update is negligible compared with that of the x-update.

Convergence of ADMM for a nonconvex problem such as this one is not guaranteed; and even when it does converge, the final result can depend on the choice of ρ and the initial values for z and u. To explore this, we ran 100 ADMM simulations with randomly chosen initial values and ρ ranging between 0.1 and 100. Indeed, some of them did not converge, or at least, were converging slowly. But most of them converged, though not to exactly the same points. However, the objective values obtained by those that converged were reasonably close to each other, typically within 5%. The different values of x found had small

variations in support (choice of regressors) and value (weights), but the largest weights were consistently assigned to the same regressors.

We now compare the use of nonconvex regressor selection with the lasso, in terms of obtaining the sparsity-fit trade-off. We obtain this curve for regressor selection by running ADMM for each value of c between $c = 1$ and $c = 120$. For the lasso, we compute the regularization path for 300 values of λ; for each x^{lasso} found, we then perform a least squares fit using the sparsity pattern in x^{lasso} to get our final x. For each cardinality, we plot the best fit found among all such x. Figure 11.8 shows the trade-off curves obtained by regressor selection and the lasso, with and without posterior least squares fit. We see that while the results are not exactly the same, they are quite similar, and for all practical purposes, equivalent. This suggests that regressor selection via ADMM can be used as well as lasso for obtaining a good cardinality-fit trade-off; it might have an advantage when the desired cardinality is known ahead of time.

12

Conclusions

We have discussed ADMM and illustrated its applicability to distributed convex optimization in general and many problems in statistical machine learning in particular. We argue that ADMM can serve as a good general-purpose tool for optimization problems arising in the analysis and processing of modern massive datasets. Much like gradient descent and the conjugate gradient method are standard tools of great use when optimizing smooth functions on a single machine, ADMM should be viewed as an analogous tool in the distributed regime.

ADMM sits at a higher level of abstraction than classical optimization algorithms like Newton's method. In such algorithms, the base operations are low-level, consisting of linear algebra operations and the computation of gradients and Hessians. In the case of ADMM, the base operations include solving small convex optimization problems (which in some cases can be done via a simple analytical formula). For example, when applying ADMM to a very large model fitting problem, each update reduces to a (regularized) model fitting problem on a smaller dataset. These subproblems can be solved using any standard serial algorithm suitable for small to medium sized problems. In this sense, ADMM builds on existing algorithms for single machines, and so can be

viewed as a modular coordination algorithm that 'incentivizes' a set of simpler algorithms to collaborate to solve much larger global problems together than they could on their own. Alternatively, it can be viewed as a simple way of 'bootstrapping' specialized algorithms for small to medium sized problems to work on much larger problems than would otherwise be possible.

We emphasize that for any particular problem, it is likely that another method will perform better than ADMM, or that some variation on ADMM will substantially improve performance. However, a simple algorithm derived from basic ADMM will often offer performance that is at least comparable to very specialized algorithms (even in the serial setting), and in most cases, the simple ADMM algorithm will be efficient enough to be useful. In a few cases, ADMM-based methods actually turn out to be state-of-the-art even in the serial regime. Moreover, ADMM has the benefit of being extremely simple to implement, and it maps onto several standard distributed programming models reasonably well.

ADMM was developed over a generation ago, with its roots stretching far in advance of the Internet, distributed and cloud computing systems, massive high-dimensional datasets, and the associated large-scale applied statistical problems. Despite this, it appears to be well suited to the modern regime, and has the important benefit of being quite general in its scope and applicability.

Acknowledgments

We are very grateful to Rob Tibshirani and Trevor Hastie for encouraging us to write this review. Thanks also to Alexis Battle, Dimitri Bertsekas, Danny Bickson, Tom Goldstein, Dimitri Gorinevsky, Daphne Koller, Vicente Malave, Stephen Oakley, and Alex Teichman for helpful comments and discussions. Yang Wang and Matt Kraning helped in developing ADMM for the sharing and exchange problems, and Arezou Keshavarz helped work out ADMM for generalized additive models. We thank Georgios Giannakis and Alejandro Ribeiro for pointing out some very relevant references that we had missed in an earlier version. We thank John Duchi for a very careful reading of the manuscript and for suggestions that greatly improved it.

Support for this work was provided in part by AFOSR grant FA9550-09-0130 and NASA grant NNX07AEIIA. Neal Parikh was supported by the Cortlandt and Jean E. Van Rensselaer Engineering Fellowship from Stanford University and by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0645962. Eric Chu was supported by the Pan Wen-Yuan Foundation Scholarship.

A

Convergence Proof

The basic convergence result given in §3.2 can be found in several references, such as [81, 63]. Many of these give more sophisticated results, with more general penalties or inexact minimization. For completeness, we give a proof here.

We will show that if f and g are closed, proper, and convex, and the Lagrangian L_0 has a saddle point, then we have primal residual convergence, meaning that $r^k \to 0$, and objective convergence, meaning that $p^k \to p^\star$, where $p^k = f(x^k) + g(z^k)$. We will also see that the dual residual $s^k = \rho A^T B(z^k - z^{k-1})$ converges to zero.

Let $(x^\star, z^\star, y^\star)$ be a saddle point for L_0, and define

$$V^k = (1/\rho)\|y^k - y^\star\|_2^2 + \rho\|B(z^k - z^\star)\|_2^2,$$

We will see that V^k is a *Lyapunov function* for the algorithm, *i.e.*, a nonnegative quantity that decreases in each iteration. (Note that V^k is unknown while the algorithm runs, since it depends on the unknown values $z^\star$ and $y^\star$.)

We first outline the main idea. The proof relies on three key inequalities, which we will prove below using basic results from convex analysis

along with simple algebra. The first inequality is

$$V^{k+1} \leq V^k - \rho\|r^{k+1}\|_2^2 - \rho\|B(z^{k+1} - z^k)\|_2^2. \tag{A.1}$$

This states that V^k decreases in each iteration by an amount that depends on the norm of the residual and on the change in z over one iteration. Because $V^k \leq V^0$, it follows that y^k and Bz^k are bounded. Iterating the inequality above gives that

$$\rho\sum_{k=0}^{\infty}\left(\|r^{k+1}\|_2^2 + \|B(z^{k+1} - z^k)\|_2^2\right) \leq V^0,$$

which implies that $r^k \to 0$ and $B(z^{k+1} - z^k) \to 0$ as $k \to \infty$. Multiplying the second expression by ρA^T shows that the dual residual $s^k = \rho A^T B(z^{k+1} - z^k)$ converges to zero. (This shows that the stopping criterion (3.12), which requires the primal and dual residuals to be small, will eventually hold.)

The second key inequality is

$$\begin{aligned} &p^{k+1} - p^\star \\ &\quad \leq -(y^{k+1})^T r^{k+1} - \rho(B(z^{k+1} - z^k))^T(-r^{k+1} + B(z^{k+1} - z^\star)), \end{aligned} \tag{A.2}$$

and the third inequality is

$$p^\star - p^{k+1} \leq y^{\star T} r^{k+1}. \tag{A.3}$$

The righthand side in (A.2) goes to zero as $k \to \infty$, because $B(z^{k+1} - z^\star)$ is bounded and both r^{k+1} and $B(z^{k+1} - z^k)$ go to zero. The righthand side in (A.3) goes to zero as $k \to \infty$, since r^k goes to zero. Thus we have $\lim_{k\to\infty} p^k = p^\star$, *i.e.*, objective convergence.

Before giving the proofs of the three key inequalities, we derive the inequality (3.11) mentioned in our discussion of stopping criterion from the inequality (A.2). We simply observe that $-r^{k+1} + B(z^{k+1} - z^k) = -A(x^{k+1} - x^\star)$; substituting this into (A.2) yields (3.11),

$$p^{k+1} - p^\star \leq -(y^{k+1})^T r^{k+1} + (x^{k+1} - x^\star)^T s^{k+1}.$$

Proof of inequality (A.3)

Since $(x^\star, z^\star, y^\star)$ is a saddle point for L_0, we have

$$L_0(x^\star, z^\star, y^\star) \leq L_0(x^{k+1}, z^{k+1}, y^\star).$$

Using $Ax^\star + Bz^\star = c$, the lefthand side is $p^\star$. With $p^{k+1} = f(x^{k+1}) + g(z^{k+1})$, this can be written as

$$p^\star \leq p^{k+1} + y^{\star T} r^{k+1},$$

which gives (A.3).

Proof of inequality (A.2)

By definition, x^{k+1} minimizes $L_\rho(x, z^k, y^k)$. Since f is closed, proper, and convex it is subdifferentiable, and so is L_ρ. The (necessary and sufficient) optimality condition is

$$0 \in \partial L_\rho(x^{k+1}, z^k, y^k) = \partial f(x^{k+1}) + A^T y^k + \rho A^T (Ax^{k+1} + Bz^k - c).$$

(Here we use the basic fact that the subdifferential of the sum of a subdifferentiable function and a differentiable function with domain $\mathbf{R}^n$ is the sum of the subdifferential and the gradient; see, *e.g.*, [140, §23].)

Since $y^{k+1} = y^k + \rho r^{k+1}$, we can plug in $y^k = y^{k+1} - \rho r^{k+1}$ and rearrange to obtain

$$0 \in \partial f(x^{k+1}) + A^T (y^{k+1} - \rho B(z^{k+1} - z^k)).$$

This implies that x^{k+1} minimizes

$$f(x) + (y^{k+1} - \rho B(z^{k+1} - z^k))^T Ax.$$

A similar argument shows that z^{k+1} minimizes $g(z) + y^{(k+1)T} Bz$. It follows that

$$\begin{aligned} &f(x^{k+1}) + (y^{k+1} - \rho B(z^{k+1} - z^k))^T Ax^{k+1} \\ &\quad \leq f(x^\star) + (y^{k+1} - \rho B(z^{k+1} - z^k))^T Ax^\star \end{aligned}$$

and that

$$g(z^{k+1}) + y^{(k+1)T} Bz^{k+1} \leq g(z^\star) + y^{(k+1)T} Bz^\star.$$

Adding the two inequalities above, using $Ax^\star + Bz^\star = c$, and rearranging, we obtain (A.2).

Proof of inequality (A.1)

Adding (A.2) and (A.3), regrouping terms, and multiplying through by 2 gives

$$\begin{aligned} 2(y^{k+1}-y^\star)^T r^{k+1} - 2\rho(B(z^{k+1}-z^k))^T r^{k+1} \\ + 2\rho(B(z^{k+1}-z^k))^T(B(z^{k+1}-z^\star)) \leq 0. \end{aligned} \tag{A.4}$$

The result (A.1) will follow from this inequality after some manipulation and rewriting.

We begin by rewriting the first term. Substituting $y^{k+1} = y^k + \rho r^{k+1}$ gives

$$2(y^k - y^\star)^T r^{k+1} + \rho\|r^{k+1}\|_2^2 + \rho\|r^{k+1}\|_2^2,$$

and substituting $r^{k+1} = (1/\rho)(y^{k+1} - y^k)$ in the first two terms gives

$$(2/\rho)(y^k - y^\star)^T(y^{k+1} - y^k) + (1/\rho)\|y^{k+1} - y^k\|_2^2 + \rho\|r^{k+1}\|_2^2.$$

Since $y^{k+1} - y^k = (y^{k+1} - y^\star) - (y^k - y^\star)$, this can be written as

$$(1/\rho)\left(\|y^{k+1} - y^\star\|_2^2 - \|y^k - y^\star\|_2^2\right) + \rho\|r^{k+1}\|_2^2. \tag{A.5}$$

We now rewrite the remaining terms, *i.e.*,

$$\rho\|r^{k+1}\|_2^2 - 2\rho(B(z^{k+1}-z^k))^T r^{k+1} + 2\rho(B(z^{k+1}-z^k))^T(B(z^{k+1}-z^\star)),$$

where $\rho\|r^{k+1}\|_2^2$ is taken from (A.5). Substituting

$$z^{k+1} - z^\star = (z^{k+1} - z^k) + (z^k - z^\star)$$

in the last term gives

$$\begin{aligned} \rho\|r^{k+1} - B(z^{k+1} - z^k)\|_2^2 + \rho\|B(z^{k+1} - z^k)\|_2^2 \\ +2\rho(B(z^{k+1} - z^k))^T(B(z^k - z^\star)), \end{aligned}$$

and substituting

$$z^{k+1} - z^k = (z^{k+1} - z^\star) - (z^k - z^\star)$$

in the last two terms, we get

$$\rho\|r^{k+1} - B(z^{k+1} - z^k)\|_2^2 + \rho\left(\|B(z^{k+1} - z^\star)\|_2^2 - \|B(z^k - z^\star)\|_2^2\right).$$

With the previous step, this implies that (A.4) can be written as

$$V^k - V^{k+1} \geq \rho \|r^{k+1} - B(z^{k+1} - z^k)\|_2^2. \tag{A.6}$$

To show (A.1), it now suffices to show that the middle term $-2\rho r^{(k+1)T}(B(z^{k+1} - z^k))$ of the expanded right hand side of (A.6) is positive. To see this, recall that z^{k+1} minimizes $g(z) + y^{(k+1)T}Bz$ and z^k minimizes $g(z) + y^{kT}Bz$, so we can add

$$g(z^{k+1}) + y^{(k+1)T}Bz^{k+1} \leq g(z^k) + y^{(k+1)T}Bz^k$$

and

$$g(z^k) + y^{kT}Bz^k \leq g(z^{k+1}) + y^{kT}Bz^{k+1}$$

to get that

$$(y^{k+1} - y^k)^T(B(z^{k+1} - z^k)) \leq 0.$$

Substituting $y^{k+1} - y^k = \rho r^{k+1}$ gives the result, since $\rho > 0$.

References

[1] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo, "Fast image recovery using variable splitting and constrained optimization," *IEEE Transactions on Image Processing*, vol. 19, no. 9, pp. 2345–2356, 2010.

[2] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo, "An Augmented Lagrangian Approach to the Constrained Optimization Formulation of Imaging Inverse Problems," *IEEE Transactions on Image Processing*, vol. 20, pp. 681–695, 2011.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson, *LAPACK: A portable linear algebra library for high-performance computers*. IEEE Computing Society Press, 1990.

[4] K. J. Arrow and G. Debreu, "Existence of an equilibrium for a competitive economy," *Econometrica: Journal of the Econometric Society*, vol. 22, no. 3, pp. 265–290, 1954.

[5] K. J. Arrow, L. Hurwicz, and H. Uzawa, *Studies in Linear and Nonlinear Programming*. Stanford University Press: Stanford, 1958.

[6] K. J. Arrow and R. M. Solow, "Gradient methods for constrained maxima, with weakened assumptions," in *Studies in Linear and Nonlinear Programming*, (K. J. Arrow, L. Hurwicz, and H. Uzawa, eds.), Stanford University Press: Stanford, 1958.

[7] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont, "Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data," *Journal of Machine Learning Research*, vol. 9, pp. 485–516, 2008.

[8] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, "Convexity, classification, and risk bounds," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156, 2006.

[9] H. H. Bauschke and J. M. Borwein, "Dykstra's alternating projection algorithm for two sets," *Journal of Approximation Theory*, vol. 79, no. 3, pp. 418–443, 1994.

[10] H. H. Bauschke and J. M. Borwein, "On projection algorithms for solving convex feasibility problems," *SIAM Review*, vol. 38, no. 3, pp. 367–426, 1996.

[11] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[12] S. Becker, J. Bobin, and E. J. Candès, "NESTA: A fast and accurate first-order method for sparse recovery," Available at `http://www.acm.caltech.edu/~emmanuel/papers/NESTA.pdf`, 2009.

[13] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, pp. 238–252, 1962.

[14] A. Bensoussan, J.-L. Lions, and R. Temam, "Sur les méthodes de décomposition, de décentralisation et de coordination et applications," *Methodes Mathematiques de l'Informatique*, pp. 133–257, 1976.

[15] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[16] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, second ed., 1999.

[17] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[18] J. M. Bioucas-Dias and M. A. T. Figueiredo, "Alternating Direction Algorithms for Constrained Sparse Regression: Application to Hyperspectral Unmixing," *arXiv:1002.4527*, 2010.

[19] J. Borwein and A. Lewis, *Convex Analysis and Nonlinear Optimization: Theory and Examples*. Canadian Mathematical Society, 2000.

[20] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[21] L. M. Bregman, "Finding the common point of convex sets by the method of successive projections," *Proceedings of the USSR Academy of Sciences*, vol. 162, no. 3, pp. 487–490, 1965.

[22] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 3, pp. 200–217, 1967.

[23] H. Brézis, *Opérateurs Maximaux Monotones et Semi-Groupes de Contractions dans les Espaces de Hilbert*. North-Holland: Amsterdam, 1973.

[24] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[25] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters," *Proceedings of the 36th International Conference on Very Large Databases*, 2010.

[26] R. H. Byrd, P. Lu, and J. Nocedal, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific and Statistical Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[27] E. J. Candès and Y. Plan, "Near-ideal model selection by ℓ_1 minimization," *Annals of Statistics*, vol. 37, no. 5A, pp. 2145–2177, 2009.

[28] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, no. 2, p. 489, 2006.

[29] E. J. Candès and T. Tao, "Near-optimal signal recovery from random projections: Universal encoding strategies?," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5406–5425, 2006.

[30] Y. Censor and S. A. Zenios, "Proximal minimization algorithm with D-functions," *Journal of Optimization Theory and Applications*, vol. 73, no. 3, pp. 451–464, 1992.

[31] Y. Censor and S. A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, 1997.

[32] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "BigTable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008.

[33] G. Chen and M. Teboulle, "A proximal-based decomposition method for convex minimization problems," *Mathematical Programming*, vol. 64, pp. 81–101, 1994.

[34] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Review*, vol. 43, pp. 129–159, 2001.

[35] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software*, vol. 35, no. 3, p. 22, 2008.

[36] W. Cheney and A. A. Goldstein, "Proximity maps for convex sets," *Proceedings of the American Mathematical Society*, vol. 10, no. 3, pp. 448–450, 1959.

[37] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "MapReduce for machine learning on multicore," in *Advances in Neural Information Processing Systems*, 2007.

[38] J. F. Claerbout and F. Muir, "Robust modeling with erratic data," *Geophysics*, vol. 38, p. 826, 1973.

[39] P. L. Combettes, "The convex feasibility problem in image recovery," *Advances in Imaging and Electron Physics*, vol. 95, pp. 155–270, 1996.

[40] P. L. Combettes and J. C. Pesquet, "A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery," *IEEE Journal on Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 564–574, 2007.

[41] P. L. Combettes and J. C. Pesquet, "Proximal Splitting Methods in Signal Processing," *arXiv:0912.3522*, 2009.

[42] P. L. Combettes and V. R. Wajs, "Signal recovery by proximal forward-backward splitting," *Multiscale Modeling and Simulation*, vol. 4, no. 4, pp. 1168–1200, 2006.

[43] G. B. Dantzig, *Linear Programming and Extensions*. RAND Corporation, 1963.

[44] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, pp. 101–111, 1960.

[45] I. Daubechies, M. Defrise, and C. D. Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, vol. 57, pp. 1413–1457, 2004.

[46] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[47] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM: Philadelphia, PA, 1997.

[48] A. P. Dempster, "Covariance selection," *Biometrics*, vol. 28, no. 1, pp. 157–175, 1972.

[49] D. L. Donoho, "De-noising by soft-thresholding," *IEEE Transactions on Information Theory*, vol. 41, pp. 613–627, 1995.

[50] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[51] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, p. 18914, 2009.

[52] D. L. Donoho and Y. Tsaig, "Fast solution of ℓ_1-norm minimization problems when the solution may be sparse," Tech. Rep., Stanford University, 2006.

[53] J. Douglas and H. H. Rachford, "On the numerical solution of heat conduction problems in two and three space variables," *Transactions of the American Mathematical Society*, vol. 82, pp. 421–439, 1956.

[54] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Distributed Dual Averaging in Networks," in *Advances in Neural Information Processing Systems*, 2010.

[55] J. C. Duchi, S. Gould, and D. Koller, "Projected subgradient methods for learning sparse Gaussians," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2008.

[56] R. L. Dykstra, "An algorithm for restricted least squares regression," *Journal of the American Statistical Association*, vol. 78, pp. 837–842, 1983.

[57] J. Eckstein, *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, MIT, 1989.

[58] J. Eckstein, "Nonlinear proximal point algorithms using Bregman functions, with applications to convex programming," *Mathematics of Operations Research*, pp. 202–226, 1993.

[59] J. Eckstein, "Parallel alternating direction multiplier decomposition of convex programs," *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 39–62, 1994.

[60] J. Eckstein, "Some saddle-function splitting methods for convex programming," *Optimization Methods and Software*, vol. 4, no. 1, pp. 75–83, 1994.

[61] J. Eckstein, "A practical general approximation criterion for methods of multipliers based on Bregman distances," *Mathematical Programming*, vol. 96, no. 1, pp. 61–86, 2003.

[62] J. Eckstein and D. P. Bertsekas, "An alternating direction method for linear programming," Tech. Rep., MIT, 1990.

[63] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, pp. 293–318, 1992.

[64] J. Eckstein and M. C. Ferris, "Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control," *INFORMS Journal on Computing*, vol. 10, pp. 218–235, 1998.

[65] J. Eckstein and M. Fukushima, "Some reformulations and applications of the alternating direction method of multipliers," *Large Scale Optimization: State of the Art*, pp. 119–138, 1993.

[66] J. Eckstein and B. F. Svaiter, "A family of projective splitting methods for the sum of two maximal monotone operators," *Mathematical Programming*, vol. 111, no. 1-2, p. 173, 2008.

[67] J. Eckstein and B. F. Svaiter, "General projective splitting methods for sums of maximal monotone operators," *SIAM Journal on Control and Optimization*, vol. 48, pp. 787–811, 2009.

[68] E. Esser, "Applications of Lagrangian-based alternating direction methods and connections to split Bregman," *CAM report*, vol. 9, p. 31, 2009.

[69] H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Research*, vol. 11, no. 3, pp. 399–417, 1963.

[70] M. J. Fadili and J. L. Starck, "Monotone operator splitting for optimization problems in sparse recovery," *IEEE ICIP*, 2009.

[71] A. V. Fiacco and G. P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. Society for Industrial and Applied Mathematics, 1990. First published in 1968 by Research Analysis Corporation.

[72] M. A. T. Figueiredo and J. M. Bioucas-Dias, "Restoration of Poissonian Images Using Alternating Direction Optimization," *IEEE Transactions on Image Processing*, vol. 19, pp. 3133–3145, 2010.

[73] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE Journal on Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 586–597, 2007.

[74] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[75] M. Fortin and R. Glowinski, *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[76] M. Fortin and R. Glowinski, "On decomposition-coordination methods using an augmented Lagrangian," in *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*, (M. Fortin and R. Glowinski, eds.), North-Holland: Amsterdam, 1983.

[77] M. Forum, *MPI: A Message-Passing Interface Standard, version 2.2*. High-Performance Computing Center: Stuttgart, 2009.

[78] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, pp. 23–37, Springer, 1995.

[79] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, p. 432, 2008.

[80] M. Fukushima, "Application of the alternating direction method of multipliers to separable convex programming problems," *Computational Optimization and Applications*, vol. 1, pp. 93–111, 1992.

[81] D. Gabay, "Applications of the method of multipliers to variational inequalities," in *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*, (M. Fortin and R. Glowinski, eds.), North-Holland: Amsterdam, 1983.

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

[83] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library Reference Manual*. Network Theory Ltd., third ed., 2002.

[84] A. M. Geoffrion, "Generalized Benders decomposition," *Journal of Optimization Theory and Applications*, vol. 10, no. 4, pp. 237–260, 1972.

[85] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[86] R. Glowinski and A. Marrocco, "Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares," *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, vol. 9, pp. 41–76, 1975.

[87] R. Glowinski and P. L. Tallec, "Augmented Lagrangian methods for the solution of variational problems," Tech. Rep. 2965, University of Wisconsin-Madison, 1987.

[88] T. Goldstein and S. Osher, "The split Bregman method for ℓ_1 regularized problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 323–343, 2009.

[89] E. G. Gol'shtein and N. V. Tret'yakov, "Modified Lagrangians in convex programming and their generalizations," *Point-to-Set Maps and Mathematical Programming*, pp. 86–97, 1979.

[90] G. H. Golub and C. F. van Loan, *Matrix Computations*. Johns Hopkins University Press, third ed., 1996.

[91] D. Gregor and A. Lumsdaine, "The Parallel BGL: A generic library for distributed graph computations," *Parallel Object-Oriented Scientific Computing*, 2005.

[92] A. Halevy, P. Norvig, and F. Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intelligent Systems*, vol. 24, no. 2, 2009.

[93] K. B. Hall, S. Gilpin, and G. Mann, "MapReduce/BigTable for distributed optimization," in *Neural Information Processing Systems: Workshop on Learning on Cores, Clusters, and Clouds*, 2010.

[94] T. Hastie and R. Tibshirani, *Generalized Additive Models*. Chapman & Hall, 1990.

[95] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, second ed., 2009.

[96] B. S. He, H. Yang, and S. L. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *Journal of Optimization Theory and Applications*, vol. 106, no. 2, pp. 337–356, 2000.

[97] M. R. Hestenes, "Multiplier and gradient methods," *Journal of Optimization Theory and Applications*, vol. 4, pp. 302–320, 1969.

[98] M. R. Hestenes, "Multiplier and gradient methods," in *Computing Methods in Optimization Problems*, (L. A. Zadeh, L. W. Neustadt, and A. V. Balakrishnan, eds.), Academic Press, 1969.

[99] J.-B. Hiriart-Urruty and C. Lemaréchal, *Fundamentals of Convex Analysis*. Springer, 2001.

[100] P. J. Huber, "Robust estimation of a location parameter," *Annals of Mathematical Statistics*, vol. 35, pp. 73–101, 1964.

[101] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, "ℓ_1 Trend filtering," *SIAM Review*, vol. 51, no. 2, pp. 339–360, 2009.

[102] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior-point method for large-scale ℓ_1-regularized least squares," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 606–617, 2007.

[103] K. Koh, S.-J. Kim, and S. Boyd, "An interior-point method for large-scale ℓ_1-regularized logistic regression," *Journal of Machine Learning Research*, vol. 1, no. 8, pp. 1519–1555, 2007.

[104] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[105] S. A. Kontogiorgis, *Alternating directions methods for the parallel solution of large-scale block-structured optimization problems*. PhD thesis, University of Wisconsin-Madison, 1994.

[106] S. A. Kontogiorgis and R. R. Meyer, "A variable-penalty alternating directions method for convex optimization," *Mathematical Programming*, vol. 83, pp. 29–53, 1998.

[107] L. S. Lasdon, *Optimization Theory for Large Systems*. MacMillan, 1970.

[108] J. Lawrence and J. E. Spingarn, "On fixed points of non-expansive piecewise isometric mappings," *Proceedings of the London Mathematical Society*, vol. 3, no. 3, p. 605, 1987.

[109] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979.

[110] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," *Advances in Neural Information Processing Systems*, vol. 13, 2001.

[111] J. Lin and M. Schatz, "Design Patterns for Efficient Graph Algorithms in MapReduce," in *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pp. 78–85, 2010.

[112] P. L. Lions and B. Mercier, "Splitting algorithms for the sum of two nonlinear operators," *SIAM Journal on Numerical Analysis*, vol. 16, pp. 964–979, 1979.

[113] D. C. Liu and J. Nocedal, "On the Limited Memory Method for Large Scale Optimization," *Mathematical Programming B*, vol. 45, no. 3, pp. 503–528, 1989.

[114] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "GraphLab: A New Parallel Framework for Machine Learning," in *Conference on Uncertainty in Artificial Intelligence*, 2010.

[115] Z. Lu, "Smooth optimization approach for sparse covariance selection," *SIAM Journal on Optimization*, vol. 19, no. 4, pp. 1807–1827, 2009.

[116] Z. Lu, T. K. Pong, and Y. Zhang, "An Alternating Direction Method for Finding Dantzig Selectors," `arXiv:1011.4604`, 2010.

[117] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*. Addison-Wesley: Reading, MA, 1973.

[118] J. Mairal, R. Jenatton, G. Obozinski, and F. Bach, "Network flow algorithms for structured sparsity," *Advances in Neural Information Processing Systems*, vol. 24, 2010.

[119] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 International Conference on Management of Data*, pp. 135–146, 2010.

[120] A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing, "An Augmented Lagrangian Approach to Constrained MAP Inference," in *International Conference on Machine Learning*, 2011.

[121] G. Mateos, J.-A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Transactions on Signal Processing*, vol. 58, pp. 5262–5276, Oct. 2010.

[122] P. J. McCullagh and J. A. Nelder, *Generalized Linear Models*. Chapman & Hall, 1991.

[123] N. Meinshausen and P. Bühlmann, "High-dimensional graphs and variable selection with the lasso," *Annals of Statistics*, vol. 34, no. 3, pp. 1436–1462, 2006.

[124] A. Miele, E. E. Cragg, R. R. Iver, and A. V. Levy, "Use of the augmented penalty function in mathematical programming problems, part 1," *Journal of Optimization Theory and Applications*, vol. 8, pp. 115–130, 1971.

[125] A. Miele, E. E. Cragg, and A. V. Levy, "Use of the augmented penalty function in mathematical programming problems, part 2," *Journal of Optimization Theory and Applications*, vol. 8, pp. 131–153, 1971.

[126] A. Miele, P. E. Mosely, A. V. Levy, and G. M. Coggins, "On the method of multipliers for mathematical programming problems," *Journal of Optimization Theory and Applications*, vol. 10, pp. 1–33, 1972.

[127] J.-J. Moreau, "Fonctions convexes duales et points proximaux dans un espace Hilbertien," *Reports of the Paris Academy of Sciences, Series A*, vol. 255, pp. 2897–2899, 1962.

[128] D. Mosk-Aoyama, T. Roughgarden, and D. Shah, "Fully distributed algorithms for convex optimization problems," Available at `http://theory.stanford.edu/~tim/papers/distrcvxopt.pdf`, 2007.

[129] I. Necoara and J. A. K. Suykens, "Application of a smoothing technique to decomposition in convex optimization," *IEEE Transactions on Automatic Control*, vol. 53, no. 11, pp. 2674–2679, 2008.

[130] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[131] A. Nedić and A. Ozdaglar, "Cooperative distributed multi-agent optimization," in *Convex Optimization in Signal Processing and Communications*, (D. P. Palomar and Y. C. Eldar, eds.), Cambridge University Press, 2010.

[132] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[133] Y. Nesterov, "Gradient methods for minimizing composite objective function," *CORE Discussion Paper, Catholic University of Louvain*, vol. 76, p. 2007, 2007.

[134] M. Ng, P. Weiss, and X. Yuang, "Solving Constrained Total-Variation Image Restoration and Reconstruction Problems via Alternating Direction Methods," *ICM Research Report*, Available at `http://www.optimization-online.org/DB_FILE/2009/10/2434.pdf`, 2009.

[135] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1999.

[136] H. Ohlsson, L. Ljung, and S. Boyd, "Segmentation of ARX-models using sum-of-norms regularization," *Automatica*, vol. 46, pp. 1107–1111, 2010.

[137] D. W. Peaceman and H. H. Rachford, "The numerical solution of parabolic and elliptic differential equations," *Journal of the Society for Industrial and Applied Mathematics*, vol. 3, pp. 28–41, 1955.

[138] M. J. D. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization*, (R. Fletcher, ed.), Academic Press, 1969.

[139] A. Ribeiro, I. Schizas, S. Roumeliotis, and G. Giannakis, "Kalman filtering in wireless sensor networks — Incorporating communication cost in state estimation problems," *IEEE Control Systems Magazine*, vol. 30, pp. 66–86, Apr. 2010.

[140] R. T. Rockafellar, *Convex Analysis*. Princeton University Press, 1970.

[141] R. T. Rockafellar, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, vol. 1, pp. 97–116, 1976.

[142] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM Journal on Control and Optimization*, vol. 14, p. 877, 1976.

[143] R. T. Rockafellar and R. J.-B. Wets, "Scenarios and policy aggregation in optimization under uncertainty," *Mathematics of Operations Research*, vol. 16, no. 1, pp. 119–147, 1991.

[144] R. T. Rockafellar and R. J.-B. Wets, *Variational Analysis*. Springer-Verlag, 1998.

[145] L. Rudin, S. J. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, pp. 259–268, 1992.

[146] A. Ruszczyński, "An augmented Lagrangian decomposition method for block diagonal linear programming problems," *Operations Research Letters*, vol. 8, no. 5, pp. 287–294, 1989.

[147] A. Ruszczyński, "On convergence of an augmented Lagrangian decomposition method for sparse convex optimization," *Mathematics of Operations Research*, vol. 20, no. 3, pp. 634–656, 1995.

[148] K. Scheinberg, S. Ma, and D. Goldfarb, "Sparse inverse covariance selection via alternating linearization methods," in *Advances in Neural Information Processing Systems*, 2010.

[149] I. D. Schizas, G. Giannakis, S. Roumeliotis, and A. Ribeiro, "Consensus in ad hoc WSNs with noisy links — part II: Distributed estimation and smoothing of random signals," *IEEE Transactions on Signal Processing*, vol. 56, pp. 1650–1666, Apr. 2008.

[150] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in ad hoc WSNs with noisy links — part I: Distributed estimation of deterministic signals," *IEEE Transactions on Signal Processing*, vol. 56, pp. 350–364, Jan. 2008.

[151] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[152] N. Z. Shor, *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag, 1985.

[153] J. E. Spingarn, "Applications of the method of partial inverses to convex programming: decomposition," *Mathematical Programming*, vol. 32, pp. 199–223, 1985.

[154] G. Steidl and T. Teuber, "Removing multiplicative noise by Douglas-Rachford splitting methods," *Journal of Mathematical Imaging and Vision*, vol. 36, no. 2, pp. 168–184, 2010.

[155] C. H. Teo, S. V. N. Vishwanathan, A. J. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *Journal of Machine Learning Research*, vol. 11, pp. 311–365, 2010.

[156] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1996.

[157] P. Tseng, "Applications of a splitting algorithm to decomposition in convex programming and variational inequalities.," *SIAM Journal on Control and Optimization*, vol. 29, pp. 119–138, 1991.

[158] P. Tseng, "Alternating projection-proximal methods for convex programming and variational inequalities," *SIAM Journal on Optimization*, vol. 7, pp. 951–965, 1997.

[159] P. Tseng, "A modified forward-backward splitting method for maximal monotone mappings," *SIAM Journal on Control and Optimization*, vol. 38, p. 431, 2000.

[160] J. N. Tsitsiklis, *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, 1984.

[161] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Transactions on Automatic Control*, vol. 31, no. 9, pp. 803–812, 1986.

[162] H. Uzawa, "Market mechanisms and mathematical programming," *Econometrica: Journal of the Econometric Society*, vol. 28, no. 4, pp. 872–881, 1960.

[163] H. Uzawa, "Walras' tâtonnement in the theory of exchange," *The Review of Economic Studies*, vol. 27, no. 3, pp. 182–194, 1960.

[164] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, p. 111, 1990.

[165] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, 2000.

[166] J. von Neumann, *Functional Operators, Volume 2: The Geometry of Orthogonal Spaces*. Princeton University Press: Annals of Mathematics Studies, 1950. Reprint of 1933 lecture notes.

[167] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[168] L. Walras, *Éléments d'économie politique pure, ou, Théorie de la richesse sociale*. F. Rouge, 1896.

[169] S. L. Wang and L. Z. Liao, "Decomposition method with a variable parameter for a class of monotone variational inequality problems," *Journal of Optimization Theory and Applications*, vol. 109, no. 2, pp. 415–429, 2001.

[170] T. White, *Hadoop: The Definitive Guide*. O'Reilly Press, second ed., 2010.

[171] J. M. Wooldridge, *Introductory Econometrics: A Modern Approach*. South Western College Publications, fourth ed., 2009.

[172] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[173] A. Y. Yang, A. Ganesh, Z. Zhou, S. S. Sastry, and Y. Ma, "A Review of Fast ℓ_1-Minimization Algorithms for Robust Face Recognition," *arXiv:1007.3753*, 2010.

[174] J. Yang and X. Yuan, "An inexact alternating direction method for trace norm regularized least squares problem," Available at `http://www.optimization-online.org`, 2010.

[175] J. Yang and Y. Zhang, "Alternating direction algorithms for ℓ_1-problems in compressive sensing," *Preprint*, 2009.

[176] W. Yin, S. Osher, D. Goldfarb, and J. Darbon, "Bregman iterative algorithms for ℓ_1-minimization with applications to compressed sensing," *SIAM Journal on Imaging Sciences*, vol. 1, no. 1, pp. 143–168, 2008.

[177] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[178] X. M. Yuan, "Alternating direction methods for sparse covariance selection," *Preprint*, Available at `http://www.optimization-online.org/DB_FILE/2009/09/2390.pdf`, 2009.

[179] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[180] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *Annals of Statistics*, vol. 32, no. 1, pp. 56–85, 2004.

[181] P. Zhao, G. Rocha, and B. Yu, "The composite absolute penalties family for grouped and hierarchical variable selection," *Annals of Statistics*, vol. 37, no. 6A, pp. 3468–3497, 2009.

[182] H. Zhu, A. Cano, and G. B. Giannakis, "Distributed consensus-based demodulation: algorithms and error analysis," *IEEE Transactions on Wireless Communications*, vol. 9, no. 6, pp. 2044–2054, 2010.

[183] H. Zhu, G. B. Giannakis, and A. Cano, "Distributed in-network channel decoding," *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 3970–3983, 2009.