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

Stephen Boyd* Neal Parikh† Eric Chu‡ Borja Peleato§
Jonathan Eckstein¶

WORKING DRAFT—November 19, 2010

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, training examples, or both. 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 paper, 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.

*Information Systems Laboratory, Electrical Engineering Department, Stanford University. Correspondence should be addressed to boyd@stanford.edu.

†Computer Science Department, Stanford University. Supported by a Cortlandt and Jean E. Van Rensselaer Engineering Fellowship, Stanford University, and an NSF Graduate Research Fellowship.

‡Information Systems Laboratory, Electrical Engineering Department, Stanford University.

§Information Systems Laboratory, Electrical Engineering Department, Stanford University.

¶Management Science and Information Systems Department and RUTCOR, Rutgers University.

Contents

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 tens or hundreds of millions 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 [HNP09] 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 paper 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. 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 this paper does 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 paper 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 truly large-scale distributed modern problems. For instance, we discuss ways of decomposing statistical problems both 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 a variety of standard frameworks. We also provide easily readable implementations of all the examples in this paper.

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 efficiently, or via an analytical solution, or in parallel. 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, linear and quadratic programming, and *consensus problems*, a general approach for distributed optimization.

In §6, we focus on problems involving the ℓ_1 norm, including basis pursuit, the lasso, sparse inverse covariance selection, and general ℓ_1 regularized loss minimization. It turns out that ADMM yields methods for these problems that are related to many state-of-the-art algorithms. In §7, we consider distributed methods for generic model fitting problems, including regularized regression models like the lasso and classification models like support vector machines). In §8, we consider the use of ADMM as a heuristic for solving some non-convex problems. In §9 we briefly discuss some practical implementation details, including how to implement the algorithm in frameworks suitable for cloud computing applications. Finally, in §10, we present the details of some numerical experiments.

2 Precursors

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

2.1 Dual ascent

Consider the equality-constrained convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b, \end{array} \tag{1}$$

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 (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 [BV04, §3.3] or [Roc70, §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 $y^\star$ by solving

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

provided there is only one minimizer of $L(x, y^\star)$. (This is the case if, for example, 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 a gradient method. Assuming that g is differentiable, the gradient $\nabla g(y)$ can be evaluated as follows. If $x^+ = \operatorname{argmin}_x L(x, y)$, then $\nabla g(y) = Ax^+ - b$, which is the *residual* for the equality constraint. The dual ascent method is the algorithm

$$x^{k+1} := \operatorname*{argmin}_x L(x, y^k) \tag{2}$$

$$y^{k+1} := y^k + \alpha^k (Ax^{k+1} - b), \tag{3}$$

where $\alpha^k > 0$ is a step size, and the superscript is the iteration counter. The first step (2) is an x-minimization step, and the second step (3) is a dual variable update. The dual variable y can be interpreted as a vector of prices, and the y-update step is called a *price update* or *price adjustment* step. It 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 makes sense 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 a *subgradient* of $-g$. This case requires a different choice of the step sizes 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* [Sho85].

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 important 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-minimization (2) fails, since $L(x, y)$ 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) splits into N separate problems that can be solved in parallel. Explicitly, the algorithm is

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} L_i(x_i, y^k) \tag{4}$$

$$y^{k+1} := y^k + \alpha^k (Ax^{k+1} - b). \tag{5}$$

The x-minimization step (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 (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 (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 [DW60] and Benders [Ben62] on large-scale linear programming, as well as in Dantzig's seminal book [Dan63]. The general idea of dual decomposition appears to be originally due to Everett [Eve63], and is explored in many early references [Las70, Geo72, Lue73, BLT76]. The use of nondifferentiable optimization, such as the subgradient method, to solve the dual problem is discussed by Shor [Sho85]. Good references on dual methods and decomposition include the book by Bertsekas [Ber99, chapter 6] and the survey by Nedić and Ozdaglar [NO10] on distributed optimization, which discusses dual decomposition methods and consensus problems.

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 [Tsi84, TBA86, BT89]. There has also been some recent work on problems where each agent has its own convex, potentially non-differentiable, objective function [NO09]. Some good reference books on parallel optimization include those by Bertsekas and Tsitsiklis [BT89] and Censor and Zenios [CZ97].

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 (1) is

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2, \tag{6}$$

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

It can be shown that g_ρ is 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 residual. Applying dual ascent to the modified problem yields the algorithm

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, y^k) \tag{7}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} - b), \tag{8}$$

which is known as the *method of multipliers* for solving (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 (8). For simplicity, we assume here that f is differentiable, though this is not required for the algorithm to work. The optimality conditions for (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

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

where the gradients are with respect to x. In other words, using ρ as the step size in the dual update makes (x^{k+1}, y^{k+1}) 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 (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.

Augmented Lagrangians and the method of multipliers for constrained optimization were first proposed in the late 1960s by Hestenes [Hes69a, Hes69b] and Powell [Pow69]. Many of the early numerical experiments on the method of multipliers are due to Miele et al. [MCIL71, MCL71, MMLC72]. Much of the early work is consolidated in a monograph by Bertsekas [Ber82], who also discusses similarities to older approaches using Lagrangians and penalty functions [AS58, AHU58, FM90], 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

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array} \tag{9}$$

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 with the general linear equality constrained problem (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 (9) 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

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \tag{10}$$

$$z^{k+1} := \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k) \tag{11}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \tag{12}$$

where $\rho > 0$. The algorithm is very similar to dual ascent and the method of multipliers: it consists of an x-minimization step (10), a z-minimization step (11), and a dual variable update (12). 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 solving the problem (9) 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 [GvL96, §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 (relabel) x and z, f and g, and A and B in the problem (9), we obtain a variation on ADMM with the order of the x-update step (10) and z-update step (11) reversed. The roles of x and z are almost symmetric, but not quite, since the dual update is done after the z-update and before the x-update.

Scaled form. ADMM can be written in a slightly different form, which is sometimes 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

$$y^T r + (\rho/2)\|r\|_2^2 = (\rho/2)\|r + (1/\rho)y\|_2^2 - (1/2\rho)\|y\|_2^2.$$

Scaling the dual variable to $u = (1/\rho)y$, we can express ADMM as

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|Ax + Bz^k - c + u^k\|_2^2\right) && (13)\\ z^{k+1} &:= \operatorname*{argmin}_z \left(g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + u^k\|_2^2\right) && (14)\\ u^{k+1} &:= u^k + (Ax^{k+1} + Bz^{k+1} - c). && (15) \end{aligned}$$

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 (10–12), the *unscaled form*, and the second form (13–15) the *scaled form*, since it is expressed in terms of a scaled version of the dual variable. The two are clearly equivalent, but one or the other may be more aesthetically appealing for a particular application.

3.2 Convergence

There are many convergence results for ADMM discussed in the literature; here, we limit ourselves to a basic but still quite 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 (9).

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 (10) and z-update (11) 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 (10) 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

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

for all $x,\ z,\ y$. This is the same as assuming that strong duality holds, and in particular that $(x^\star, z^\star)$ are primal optimal and $y^\star$ is dual optimal. (Strong duality holds for most convex problems; see [BV04, §5.2.3, §5.4.1–§5.4.2] for background.) 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 (9), so $Ax^\star + Bz^\star = c$ and $f(x^\star) < \infty$, $g(z^\star) < \infty$. It also implies that $y^\star$ is dual optimal. 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.

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 this result 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.

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 a 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, 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 [EF98], 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.

3.3 Optimality conditions and stopping criterion

The necessary and sufficient optimality conditions for the ADMM problem (9) are primal feasibility,

$$Ax^\star + Bz^\star - c = 0, \tag{16}$$

and dual feasibility,

$$\begin{aligned} 0 &\in \partial f(x^\star) + A^T y^\star & (17)\\ 0 &\in \partial g(z^\star) + B^T y^\star. & (18) \end{aligned}$$

Here, ∂ denotes the subdifferential operator. (When f and g are differentiable, ∂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 (18), so attaining optimality comes down to satisfying (16) and (17). This phenomenon is analogous to the iterates of the method of multipliers always being dual feasible; see page 9.

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 (17). In the sequel, we will refer to s^{k+1} as the *dual residual* at iteration $k+1$. (We refer 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, (16)–(18). The last condition (18) always holds for $(x^{k+1}, z^{k+1}, y^{k+1})$; the residuals for the other two, (16) and (17), 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.)

Stopping criteria. The discussion above suggests that a reasonable termination criterion is that the primal and dual residuals be small, *i.e.*,

$$\|r^k\|_2 \leq \epsilon^{\text{pri}} \quad \text{and} \quad \|s^k\|_2 \leq \epsilon^{\text{dual}}, \tag{19}$$

where $\epsilon^{\text{pri}} > 0$ and $\epsilon^{\text{dual}} > 0$ are feasibility tolerances for the primal and dual feasibility conditions (16) and (17), 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.

The stopping criterion (19) is in terms of the residuals of the optimality conditions. We can relate these residuals 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

$$f(x^k) + g(z^k) - p^\star \leq -(y^k)^T r^k + (x^k - x^\star)^T s^k. \tag{20}$$

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).

3.4 Extensions and variations

Many variations on the classic ADMM algorithm have been explored in the literature. Here we briefly survey a few 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).

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 [Roc76b], 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 usually works well is the following (see, *e.g.*, [HYW00, WL01]):

$$\rho^{k+1} := \begin{cases} \tau^{\text{incr}}\rho^k & \text{if } \|r^k\|_2 > \mu\|s^k\|_2 \\ \rho^k/\tau^{\text{decr}} & \text{if } \|s^k\|_2 > \mu\|r^k\|_2 \\ \rho^k & \text{otherwise,} \end{cases} \tag{21}$$

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 (21) 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.

More general augmenting terms. Another idea is to allow for a different penalty parameter for each constraint, or more generally, replacing the quadratic term $(\rho/2)\|r\|_2^2$ with $(1/2)r^TPr$, 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 $F^TF = P$.

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 [EB92], and numerical experiments in [Eck94a, EF98] suggest that over-relaxation with $\alpha^k \in [1.5, 1.8]$ can improve convergence.

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 [EB92], building on earlier results by Gol'shtein and Tret'yakov [GT79]. This technique is important in situations where the x- or z-minimizations 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.

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.*, [Rus89]. 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 [BT89, §3.4.4]. Another variation is to perform an additional y-update between the x- and z-update, with half the step length [BT89].

Related algorithms. There are also a number of other algorithms distinct from but inspired by ADMM. For instance, Fukushima [Fuk92] applies ADMM to a dual problem formulation, yielding a 'dual ADMM' algorithm, which is shown in [EF93] to be equivalent to the 'primal Douglas-Rachford' method discussed in [Eck89, §3.5.6]. There are also algorithms resembling a combination of ADMM and the *proximal* method of multipliers [Roc76a], rather than the standard method of multipliers; see, *e.g.*, [CT94, Eck94b]. Other representative publications include [EB90, RW91, Eck94a, Rus95, Tse97, Tse00, CW06].

3.5 Notes and references

ADMM was originally proposed in the mid-1970s by Glowinski and Marrocco [GM75] and Gabay and Mercier [GM76]. There are a number of other important papers analyzing the properties of the algorithm, including [FG83b, Gab83, FG83a, GT87, Tse91, Fuk92, EF93, CT94]. In particular, the convergence of ADMM has been explored by many authors, including Gabay [Gab83], and Eckstein and Bertsekas [EB92].

ADMM has also been applied to a number of statistical problems, such as constrained sparse regression [BDF10], sparse signal recovery [FS09], image restoration and denoising [FBD10, ST10, NWY09], trace norm-regularized least squares minimization [YY10], and support vector machines [FCG10], among others. For examples in signal processing, see [CW06, CP07, CP09] and the references therein.

Many papers analyzing ADMM do so from the perspective of *maximal monotone operators* [Bre73, Roc76a, Roc76b, EB92, RW98]. 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* [Roc76b] 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 [Eck89, §2].

The method of multipliers was shown to be a special case of the proximal point algorithm by Rockafellar [Roc76a]. Gabay [Gab83] showed that ADMM is a special case of a method called *Douglas-Rachford splitting* for monotone operators [DR56, LM79], and Eckstein and Bertsekas [EB92] 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* [PR55, LM79] instead, as shown by Glowinski and Le Tallec [GT87].) Using the same framework, Eckstein and Bertsekas [EB92] also showed the relationships between a number of other algorithms, such as Spingarn's method of partial inverses [Spi85]. Lawrence and Spingarn [LS87] develop an alternative framework showing that Douglas-Rachford splitting, hence ADMM, is a special case of the proximal point algorithm; Eckstein and Ferris [EF98] offer a more recent discussion explaining this conceptual framework.

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 [ES08, ES09].

A considerable body of recent research considers replacing quadratic penalty term in the standard method of multipliers with a more general deviation penalty, such as one derived from a *Bregman divergence* [CZ92, Eck93]; see [Bre67] 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

Depending on the form of f and g, it is sometimes possible to express the x- and z-updates in a simplified form that can be exploited to carry out these iterations more efficiently; in some cases, the x- or z-updates can each split into a set of smaller operations that can be carried out in parallel. We will now discuss some general patterns that recur throughout the applications. The examples will be written for the x-update but apply 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

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

Viewed 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 [Mor62]. In variational analysis,

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

is known as the *Moreau envelope* or *Moreau-Yosida regularization* of f, and is connected to the theory of the proximal point algorithm [RW98]. 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 [CP09] 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 (with 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 function

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 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

$$x^+ = (P + \rho A^T A)^{-1}(\rho A^T v - q). \tag{22}$$

In other words, computing the x-update amounts to solving a linear system with positive semidefinite coefficient matrix $P + \rho A^T A$ and righthand 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 [Dem97] or [GvL96]; see [BV04, appendix C] for a short overview of direct methods.

Direct methods. A *direct method* for solving a linear system $Fx = g$ is one based on first *factoring* $F = F_1 F_2 \cdots F_k$ into a product of simpler matrices, and then computing $x = F^{-1}b$ by *solving* a sequence of problems of the form $F_i z_i = z_{i-1}$, where $z_1 = F_1^{-1} g$ and $x = z_k$. The solve step is sometimes also called a *back-solve*. The cost of factorization and back-solve operations depend on the sparsity pattern of F. If the cost of factoring F is f flops, and the cost of the back-solve is s, then the total cost is $f + s$ to solve the linear system $Fx = g$. In our case, the coefficient matrix is $F = P + \rho A^T A$ and the righthand side is $g = \rho A^T v - q$, where $P \in \mathbf{S}_+^n$ and $A \in \mathbf{R}^{p \times n}$. If we exploit no structure in F, we can carry out a Cholesky factorization of F. In this case, f is $O(pn^2 + n^3)$ flops (pn^2 to form $A^T A$ and n^3 for the subsequent factorization) and s is $O(pn + n^2)$ flops (pn to form $A^T v$ and n^2 for the solve).

However, when the coefficient matrix F has special structure, 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 f and s are $O(n)$. If $P + \rho A^T A$ is banded with bandwidth k, then f is $O(pn^2 + nk^2)$ and s is $O(pn + nk)$. The same approach works when $P + \rho A^T A$ is sparse and a direct method is used, with permutations chosen to reduce fill-in.

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 we factor F once and then carry out back-solves for each righthand side. The total cost is $f + Ns$ instead of $N(f + s)$, which we would pay if we were to repeatedly factor F. So as long as ρ does not change, we can factor $P + \rho A^T A$ *once*, 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 naïve implementation, in which we solve the equations from scratch in each iteration.

When structure is exploited, the ratio between f 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.

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 (the dominant cost is forming $AP^{-1}A^T$), an improvement by a factor of $(n/p)^2$ over the naive method. The factors of $I + \rho A P^{-1} A^T$ 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^+ also makes 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 $I + \rho^k A P^{-1} A^T$ 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 a ρ varying in each iteration.

Iterative methods. Instead of direct methods, *iterative methods* such as the conjugate gradient method can be used to carry out the x-update. These methods typically only require a method for multiplying a vector by P, A, and A^T. Again, there are several ways to improve efficiency.

First, the algorithm can be initialized at the solution obtained in the previous iteration, called a *warm start*. The previous solution often gives an acceptable approximation to the update in fewer iterations than if the iterative algorithm is started at zero, the 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. The algorithm can also be terminated early, called *early stopping*, a technique that 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 repeatedly increase in each iteration.

These comments also hold when f is not quadratic, but an iterative algorithm such as the conjugate gradient method is used to approximately compute the x-minimization step.

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:

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

Here, x^+ is still an affine function of v, and the update involves solving the KKT 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 Decomposition

Suppose that $x = (x_1, \ldots, x_N)$ is a partition of the variable x into subvectors and that f is separable with respect to this partition, *i.e.*, $f(x) = f_1(x_1) + \cdots + f_N(x_N)$. 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 L_ρ is separable in x. This means that the x-minimization step can be carried out in parallel, with each subvector x_i updated by a separate minimization. It is even possible that *both* the x-minimization step and z-minimization step decompose even though all the variables are coupled in the full problem.

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. We will refer to this as the *fully separable* case. For brevity, we will sometimes abuse notation by using x_i to refer either to the ith coordinate of x_i or the ith block of the partition $(x_1, \ldots, x_N)$ depending on whether f is fully separable or just separable, but the distinction will always be clear from context and the structure of f.

As an example, 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 [Roc70, §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,

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

We refer to updates that reduce to this form as *elementwise 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

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{C}, \end{array} \tag{23}$$

where f and $\mathcal{C}$ are convex. This problem can be rewritten in ADMM form (9) 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

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

The x-update involves minimizing f plus a convex quadratic function, and the z-update is projection onto $\mathcal{C}$. The objective f need not be smooth here; indeed, we can include constraints 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 a few more specific examples.

5.1 Convex feasibility

Alternating projections. A classic problem is to find a point in the intersection of two closed nonempty convex sets $\mathcal{C}$ and $\mathcal{D}$. The classical method, which dates back to the 1930s, is von Neumann's *alternating projections* algorithm [vN50, CG59, Bre65]:

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

where $\Pi_{\mathcal{C}}$ and $\Pi_{\mathcal{D}}$ are projection onto $\mathcal{C}$ 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 the indicator function of $\mathcal{D}$. The scaled form of ADMM is then

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

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 [Dyk83, BB94], which is far more efficient than the classical method that does not use the dual variable u.

For this problem, 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 $\mathbf{dist}(\mathcal{C}, \mathcal{D})$. 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 $\mathcal{D}$, with distance no more than ϵ^{pri} from each other. Alternatively, the point $(1/2)(x^k + z^k)$ is no more than a distance $\epsilon^{\text{pri}}/2$ from both $\mathcal{C}$ and $\mathcal{D}$.

Projection on multiple sets. We can generalize this 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

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

If $x = (x_1, \ldots, x_N) \in \mathbf{R}^{nN}$, then projecting x onto $\mathcal{C}$ yields $(\Pi_{\mathcal{A}_1}(x_1), \ldots, \Pi_{\mathcal{A}_N}(x_N))$, and projecting x onto $\mathcal{D}$ yields $(\overline{x}, \overline{x}, \ldots, \overline{x})$, where $\overline{x} = (1/N)\sum_{i=1}^N x_i$. 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 §4.3, where the indicator function of $\mathcal{A}_1 \cap \cdots \cap \mathcal{A}_N$ splits into the sum of the indicator functions of each $\mathcal{A}_i$.

There is a large literature on successive projection algorithms and their many applications; see the survey by Bauschke and Borwein [BB96] for a general overview, Combettes [Com96] for applications to image processing, and Censor and Zenios [CZ97, §5] for a discussion in the context of parallel optimization.

5.2 Linear and quadratic programming

The *standard form quadratic program*

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = b, \quad x \geq 0, \end{array} \tag{24}$$

which is a linear program when $P = 0$, can be rewritten 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 quadratic objective, with domain that includes the linear equality constraints, and g is the indicator function of the nonnegative orthant $\mathbf{R}^n_+$.

The scaled form of ADMM consists of the iterations

$$\begin{array}{lcl} x^{k+1} & := & \underset{x}{\text{argmin}} \left(f(x) + (\rho/2)\|x - (z^k - u^k)\|_2^2\right) \\ z^{k+1} & := & (x^{k+1} + u^k)_+ \\ u^{k+1} & := & u^k + (x^{k+1} - z^{k+1}). \end{array}$$

The x-update is an equality-constrained least squares problem with optimality conditions

$$\left[\begin{array}{cc} P + \rho I & A^T \\ A & 0 \end{array}\right] \left[\begin{array}{c} x^{k+1} \\ \nu \end{array}\right] + \left[\begin{array}{c} q - \rho(z^k - u^k) \\ -b \end{array}\right] = 0.$$

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.

More generally, any conic constraint $x \in \mathcal{K}$ can be used in place of the constraint $x \geq 0$, in which case (24) is 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.

5.3 Global variable consensus optimization

Consensus optimization is a general form for posing and solving distributed optimization problems, and is the framework we will use for achieving parallelism in the rest of the paper. We first consider the case with a single global variable, with the objective and constraint terms split into N parts:

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

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 could say that the parameter 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 global variable z:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N. \end{array} \tag{25}$$

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, which show up frequently but do not split due to the variable being shared across terms, into separable objectives, which parallelize easily. For a useful recent discussion of consensus algorithms, see [NO10] and the references therein.

ADMM for the problem (25) can be derived either 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 (23) 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 avoid an abundance of parentheses. 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. Let the average over $i = 1, \ldots, N$ be denoted with an overline, *i.e.*, $\overline{x} = (1/N) \sum_{i=1}^N x_i$. Then the z-update can be written

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

Similarly, averaging the y-update over i 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

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT}(x_i - \overline{x}^k) + (\rho/2)\|x_i - \overline{x}^k\|_2^2\right) \tag{26}$$

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - \overline{x}^{k+1}). \tag{27}$$

This is an extremely 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.

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

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

When the original consensus problem is a parameter fitting problem, the x-update step has a natural 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.

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

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

The resulting ADMM algorithm is

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \left(f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)\|x_i - z^k\|_2^2\right) \tag{28}$$

$$z^{k+1} := \operatorname*{argmin}_{z} \left(g(z) + \sum_{i=1}^N (-y_i^{kT} z + (\rho/2)\|x_i^{k+1} - z\|_2^2)\right) \tag{29}$$

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - z^{k+1}). \tag{30}$$

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:

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

In the case with nonzero g, we do not in general have $\sum_{i=1}^N y_i^k = 0$, 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:

$$z^{k+1} := S_{\lambda/N\rho}(\tilde{z}^{k+1}).$$

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,

$$z^{k+1} := (\tilde{z}^{k+1})_+.$$

5.4 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

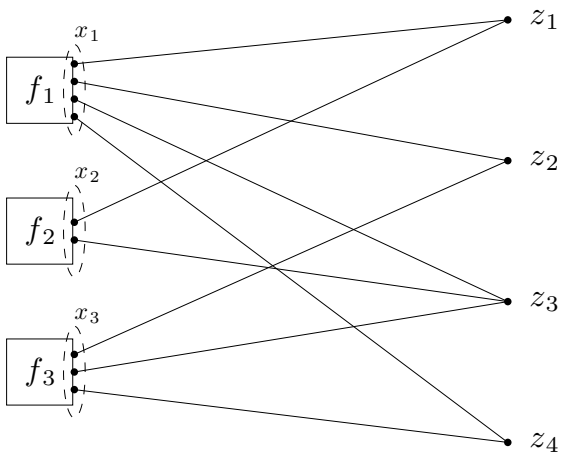

Figure 1: General form consensus optimization. Local objective terms are on the left; global variable components are on the right. Each edge in the bipartite graph is a consistency constraint, linking a local variable and a global variable component.

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.

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

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_i - \tilde{z}_i = 0, \quad i = 1, \ldots, N, \end{array} \tag{31}$$

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

A simple example is shown in Figure 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 the general form consensus problem (31) 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.

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

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + g(z) \\ \text{subject to} & x_i - \tilde{z}_i = 0, \quad i = 1, \ldots, N, \end{array} \tag{32}$$

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.

6 ℓ_1 norm problems

In previous sections, we have discussed why ADMM, due to its combined use of duality, penalty functions, and alternating minimization, is in general a more robust approach to distributed optimization than algorithms like dual decomposition that rely on subgradient methods. The problems in this section will help illustrate in turn 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 boil down to minimizing a loss function together with a regularization term or side constraints. In other cases, these side constraints are introduced through simple problem transformations like putting the problem in consensus form.

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. 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 instances of these problems for simplicity; the problem of distributed model fitting will be treated in the following section.

The idea of ℓ_1 regularization is decades old, and traces back to Huber's [Hub64] work on robust statistics and a paper of Claerbout [CM73] in geophysics. There is a vast literature, but some important modern papers are those on total variation denoising [ROF92], soft thresholding [Don95], the lasso [Tib96], basis pursuit [CDS01], compressed sensing [Don06, CRT06, CT06], and structure learning of sparse graphical models [MB06].

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. [YGZ+10] compares and benchmarks a number of representative algorithms, including gradient projection [FNW07, KKL+07], homotopy methods [DT06], iterative shrinkage-thresholding [DDM04], proximal gradient [Nes83, Nes07, BT09, BBC09], and augmented Lagrangian methods [YZ09]. There are other approaches as well, such as Bregman iterative algorithms [YOGD08] and iterative thresholding algorithms [DMM09] implementable in a message-passing framework.

6.1 Least absolute deviations

A simple variant on least squares fitting is the *least absolute deviations* problem,

$$\text{minimize} \quad \|Ax - b\|_1.$$

Least absolute deviations provides a more robust fit than least squares when the dataset contains large outliers, and has been used extensively in statistics and econometrics. See, for example, [HTF09, §10.6], [Woo09, §9.6], and [BV04, §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, ADMM can be expressed as

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

assuming A^TA is invertible. As in §4.2, the matrix A^TA can be factored once and then be used in cheaper back-solves in subsequent x-updates.

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 $m < n$. In other words, the goal is to find a sparse solution to an underdetermined system of linear equations, much like the pseudo-inverse is a solution with minimum ℓ_2 norm to an underdetermined system. This problem plays a central role in modern statistical signal processing, particularly the theory of compressed sensing; see [BDE09] for a recent introductory survey.

Basis pursuit can be written in a form suitable for ADMM 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

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

where Π is projection onto $\{x \in \mathbf{R}^n \mid Ax = b\}$. We can write the x-update explicitly as

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

and again, the comments on efficient computation from §4.2 apply.

A recent class of algorithms called *Bregman iterative methods* have attracted considerable interest for solving ℓ_1 problems like basis pursuit. It is interesting to note that for basis pursuit and related problems, *Bregman iterative regularization* [YOGD08] is equivalent to the method of multipliers, and the *split Bregman method* [GO09] is equivalent to ADMM.

6.3 General ℓ_1 regularized loss minimization

Consider the generic problem

$$\text{minimize} \quad l(x) + \lambda\|x\|_1, \tag{33}$$

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

$$\begin{array}{rcl} x^{k+1} & := & \underset{x}{\text{argmin}} \left(l(x) + y^{kT}x + (\rho/2)\|x - z^k\|_2^2\right) \\ z^{k+1} & := & S_{\lambda/\rho}(x^{k+1} + y^k/\rho) \\ y^{k+1} & := & y^k + \rho(x^{k+1} - z^{k+1}). \end{array}$$

The structure of l will dictate the complexity of the x-minimization. For example, if l is differentiable, this can be done by any standard method, such as Newton's method, a quasi-Newton method such as BFGS, or the conjugate gradient method. If f is separable, then the x-minimization will decompose, and in some cases, it may have an analytical solution.

A very wide variety of models can be represented with the loss function l, including generalized linear models [MN91] and generalized additive models [HT90]. 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.*, [HTF09, §4.4.4].

In order to use a regularizer $g(z)$ other than $\|z\|_1$, we simply need to 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 (33) is ℓ_1 regularized linear regression, also called the *lasso* [Tib96]. This involves solving

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1, \tag{34}$$

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 [HTF09, §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 [HTF09, §18.4] for a representative case study.

In ADMM form, the lasso problem can be written

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

where $f(x) = (1/2)\|Ax - b\|_2^2$ and $g(z) = \lambda\|z\|_1$. By §4.2 and §4.3, ADMM becomes

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

Note that $A^TA + \rho I$ is always invertible, assuming $\rho > 0$.

The lasso problem can be generalized to

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|Fx\|_1, \tag{35}$$

where F is an arbitrary linear transformation. An important special case is when F is the difference matrix and $A = I$, in which case this reduces to

$$\text{minimize} \quad (1/2)\|x - b\|_2^2 + \lambda\textstyle\sum_{i=1}^n |x_{i+1} - x_i|. \tag{36}$$

The second term is the *total variation* of x. This problem is often called *total variation denoising* [ROF92], and has significant applications in signal processing.

In ADMM form, the generalized lasso problem (35) can be rewritten 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

$$\begin{array}{rcl} x^{k+1} & := & (A^TA + \rho F^TF)^{-1}(A^Tb + \rho F^Tz^k - F^Ty^k) \\ z^{k+1} & := & S_{\lambda/\rho}(Fx^{k+1} + y^k/\rho) \\ y^{k+1} & := & y^k + \rho(Fx^{k+1} - z^{k+1}). \end{array}$$

For the special case of total variation denoising (36), $A^TA + \rho F^TF$ is tridiagonal, so the x-update can be carried out in $O(n)$ steps [GvL96, §4.3].

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 entries of Σ_{ij}^{-1} are 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 [KF09]. 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 since for a fixed sparsity pattern, determining the maximum likelihood estimate of Σ is a tractable 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 [BGd08]. 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 $\mathbf{S}_{++}^n$. This is is a special case of the general ℓ_1 regularized problem (33) with loss function $l(X) = \mathbf{Tr}(SX) - \log\det X$.

The idea of covariance selection is originally due to Dempster [Dem72] and was first studied in the sparse, high-dimensional regime by Meinshausen and Bühlmann [MB06]. The form of the problem above is due to Banerjee et al. [BGd08]. Some other recent papers on this problem include ones on Friedman et al.'s *graphical lasso* [FHT08], Duchi et al. [DGK08], Lu [Lu09], and Yuan [Yua09], the last of which shows that ADMM is competitive with 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.

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

$$\rho X - X^{-1} = \rho(Z^k - U^k) - S. \tag{37}$$

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 (37) 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 (37), 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 Distributed model fitting

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

$$\text{minimize} \quad l(Ax-b)+r(x), \tag{38}$$

with parameters $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m\times n}$ is the feature matrix of inputs, $b \in \mathbf{R}^m$ are the outputs, $l : \mathbf{R}^m \to \mathbf{R}$ 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 function 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. Usually, all the l_i are the same, but this assumption is not required.

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. (Many other regularization functions, like the *group lasso* [YL06] and the *elastic net* [ZH05], are just combinations of these two.) In some cases, one or more model parameters are not regularized, such as the offset parameter in a classification model. This corresponds to $r(x) = \lambda\|x_{1:n-1}\|_1$, 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_n is not regularized.

The next section discusses some examples that have the general form above. We then consider two ways to solve (38) in a distributed manner, namely, by splitting across training examples and by splitting across features.

7.1 Examples

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.*, [BV04, §7.1.1]. Then the negative log-likelihood function is $l(Ax-b)$ and $l_i(u_i) = -\log p_i(-u_i)$. If $r = 0$, then the general fitting problem (38) 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.

Classification. Many classification problems can also be put in the form of the general model fitting problem (38). We follow the standard classification setting described

in [BJM06]. Let $a_i \in \mathbf{R}^k$ denote the feature vector of the ith example and let $b_i \in \{-1, 1\}$ denote the binary outcome or class label, for $i = 1, \ldots, N$. The goal is to find a weight vector $w \in \mathbf{R}^k$ and offset $v \in \mathbf{R}$ such that

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

holds for many examples. In the literature, $a_i^T w + v$ is referred to as a *discriminant function*, viewed as a function of a_i. This condition can also be written as $u_i > 0$, where $u_i = b_i(a_i^T w + v)$ 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

$$l_i(u_i) = l_i(b_i(a_i^T w + v)).$$

A classification error is made if and only if the margin is negative, so l_i should be positive and increasing 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:

$$\frac{1}{N} \sum_{i=1}^{N} l_i(b_i(a_i^T w + v)) + r^{\mathrm{wt}}(w). \tag{39}$$

This has the generic model fitting form (38), where (w, v) corresponds to x, the a_i above is $(-b_i a_i, b_i v)$ here, the b_i above is the constant 1, and the regularizer is $r(x) = r^{\mathrm{wt}}(w)$.

In statistical learning theory, the problem (39) 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. For background, see [Vap00, Zha04, BJM06].

Many classification models in machine learning correspond to different choices of l_i and r^{wt}. Some common loss functions are *hinge loss* $(1 - u_i)_+$, *exponential loss* $\exp(-u_i)$, and *logistic loss* $\log(1 + \exp(-u_i))$; as before, the most common regularizers are ℓ_1 and ℓ_2. The *support vector machine* [SS02] corresponds to hinge loss with a quadratic penalty on w, while exponential loss yields *boosting* [FS95] and logistic loss yields logistic regression.

7.2 Splitting across examples

Here we discuss how to solve the model fitting problem (38) 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.

The solution follows immediately from the earlier discussion, by partitioning the examples into N groups, $\mathcal{D}_1, \ldots, \mathcal{D}_N$, and writing the problem in global consensus form as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) + r(z) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N, \end{array} \tag{40}$$

where

$$f_i(x) = \sum_{j \in \mathcal{D}_i} l_j(a_j^T x - b_j)$$

is the objective term associated with the ith block of examples. The problem can now be solved by applying the generic global variable consensus ADMM algorithm described in §5.3. (If appropriate, the general form consensus approach could also be used; for example, this could be useful if many features only appear in a small number of the local datasets.) In each step, each processor will solve the quadratically penalized model fitting problem

$$\text{minimize} \quad f_i(x_i) + (\rho/2)\|x_i - z^k + u_i^k\|_2^2$$

with local variable x_i. If f_i is differentiable, this can be done by any standard method, such as Newton's method, BFGS, or the conjugate gradient method. These iterative methods can be warm started for efficiency, as in §4.2; in fact, rather than making the problem more difficult, the quadratic regularization term can greatly improve the convergence rate and robustness of these algorithms compared to the problem of minimizing f_i alone.

7.3 Splitting across features

We now consider (38) 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 observed DNA mutations in each individual. There are many examples in other areas as well, and the goal is to solve problems of this kind in a distributed fashion, with each processor handling a subset of the features.

The solution involves working with a dual of (38). First, the primal problem (38) can be reformulated as

$$\begin{array}{ll} \text{minimize} & l(v) + r(x) \\ \text{subject to} & Ax - b = v, \end{array}$$

with variables $x \in \mathbf{R}^n$ and $v \in \mathbf{R}^m$, where $n \gg m$. The Lagrangian is

$$L(x, v, \nu) = l(v) + r(x) + \nu^T(Ax - b - v),$$

with dual variable $\nu \in \mathbf{R}^m$, and the dual function is

$$-l^*(\nu) - \nu^T b - r^*(-A^T\nu),$$

where l^* is the conjugate of l and r^* is the conjugate of r. The dual model fitting problem is to maximize the dual function over ν, or equivalently,

$$\text{minimize} \quad l^*(\nu) + \nu^T b + r^*(-A^T\nu), \tag{41}$$

with variable $\nu \in \mathbf{R}^m$. The objective function can take on the value $+\infty$, so this may be a constrained optimization problem. We will see later how to recover an optimal primal variable $x^\star$ from a solution of this dual problem. In the dual model fitting problem (41), the variable ν is of modest size (the number of training examples m), while the term $A^T\nu$ is large (the number of features n). Since r is separable, so is r^*, and indeed, $r_i^* = (r_i)^*$.

To split the original problem across features, partition x as $x = (x_1, \ldots, x_N)$, with $x_i \in \mathbf{R}^{n_i}$, and conformably partition the data matrix $A = [A_1 \cdots A_N]$ and the conjugate regularization function $r^*(\omega) = r_1^*(\omega_1) + \cdots + r_N^*(\omega_N)$. The dual model fitting problem (41) can then be expressed as

$$\begin{array}{ll} \text{minimize} & l^*(\nu) + \nu^T b + \sum_{i=1}^N r_i^*(-A_i^T\mu_i) \\ \text{subject to} & \mu_i = \nu, \quad i = 1, \ldots, N, \end{array} \tag{42}$$

with variables $\nu, \mu_1, \ldots, \mu_N \in \mathbf{R}^n$. This is a standard consensus problem, so ADMM has the form

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

The second step is simple since it is separable at the component level. It can be written as

$$\nu^{k+1} := \operatorname*{argmin}_{\nu} \left(l^*(\nu) + b^T\nu + (N\rho/2)\|\nu - v\|_2^2 \right),$$

where

$$v = \frac{1}{N} \sum_{i=1}^N \left(\mu_i^{k+1} + (1/\rho) y_i^k \right). \tag{43}$$

The first step can be given an interesting interpretation as a model fitting problem on its own. Comparing the form of the μ_i-update with the dual problem (41), we see that the μ_i-update involves solving the dual of the problem

$$\text{minimize} \quad (1/2\rho)\|A_i x_i - (y_i^k - \rho\nu^k)\|_2^2 + r_i(x_i).$$

This is evidently a model fitting problem involving only the features in the ith block, a quadratic loss function, and modified measurements $y_i^k - \rho\nu^k$.

Recovering the optimal model parameters. It remains to see how to get the optimal model parameters $x_i^\star$ from a solution of the dual problem. The solution $x_i^\star$ can be recovered in several ways, from $\mu_i^\star$, or from $y_i^\star$, or as a side product of the updates in ADMM.

An optimal primal value $x_i^\star$ can be obtained as the solution of the problem

$$\begin{array}{ll} \text{minimize} & r_i(x_i) \\ \text{subject to} & A_i x_i = y_i^\star, \end{array}$$

for $i = 1, \ldots, N$. For example, with Tikhonov regularization, *i.e.*, $r_i(x_i) = (\lambda/2)\|x_i\|_2^2$, we have $x_i^\star = A_i^\dagger y_i^\star$, where $A_i^\dagger$ is the pseudo-inverse of A_i.

A solution $x_i^\star$ can also be recovered from $\mu_i^\star$ when r_i is strictly convex. In this case, $x_i^\star = (\partial r_i)^{-1}(-A_i^T \mu_i)$, where ∂r_i is the subdifferential point-to-set mapping. The inverse of this relation is a function when r_i is strictly convex.

The conjugate r_i^* of the ℓ_1 penalty $r_i(x_i) = \lambda\|x_i\|_1$ is the indicator function for $-\lambda\mathbf{1} \leq x_i \leq \lambda\mathbf{1}$, the ℓ_∞-ball of radius λ. This constraint can be made explicit in the μ_i-update, so μ_i is obtained by solving the quadratic program

$$\begin{array}{ll} \text{minimize} & y_i^{kT}\mu_i + (\rho/2)\|\mu_i - \nu^k\|_2^2 \\ \text{subject to} & -\lambda\mathbf{1} \leq -A_i\mu_i \leq \lambda\mathbf{1}. \end{array} \tag{44}$$

Let χ_- be an optimal Lagrange multiplier associated with the constraint $-\lambda\mathbf{1} \leq -A_i\mu_i$, and χ_+ be an optimal Lagrange multiplier associated with the constraint $-A_i\mu_i \leq \lambda\mathbf{1}$. Then we have $x_i^\star = \chi_+ - \chi_-$.

As an example, consider the lasso problem, so $l(v) = \|v\|_2^2$ and $r(x) = \lambda\|x\|_1$. The μ_i-updates, which are carried out in parallel, involve solving the quadratic program (44); the optimal Lagrange multipliers then give the optimal model parameters once ADMM converges. The ν-update has the very simple form

$$\nu^{k+1} := \frac{1}{N\rho + 1}(N\rho v - b),$$

where v is given by (43). This step involves a gather operation to form v.

8 Nonconvex problems

This section explores the use of ADMM for *nonconvex* problems, in which case ADMM must be considered just another local optimization method, since global optimality cannot be guaranteed. 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.

8.1 Nonconvex updates

Consider the standard problem (9), but with f or g possibly nonconvex. Carrying out ADMM requires solving the x- and z-minimizations, which are not convex problems when f or g are nonconvex. In general, one could simply carry out these updates using local optimization instead of global optimization, but in a few important special cases, it is possible to perform these updates by finding the *global* solution to a nonconvex problem.

Suppose that f is convex and that g is the indicator function of a nonconvex set $\mathcal{S}$, which gives the generic constrained minimization problem (23), only with a nonconvex constraint set. For this problem, ADMM has the form

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

where $\Pi_{\mathcal{S}}$ is projection onto $\mathcal{S}$. The x-minimization step is convex since f is convex, but the z-update is projection onto a nonconvex set. In general, this is NP-Hard, but can be solved exactly in some important special cases.

- *Cardinality.* If $\mathcal{S} = \{x \mid \mathbf{card}(x) \leq k\}$, where **card** gives the number of nonzero elements, then $\Pi_{\mathcal{S}}(v)$ keeps the k largest magnitude elements and zeroes out the rest.
- *Rank.* If $\mathcal{S}$ is the set of matrices with rank k, 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 k dyads, *i.e.*, form $\Pi_{\mathcal{S}}(v) = \sum_{i=1}^k \sigma_i u_i v_i^T$.
- *Boolean constraints.* If $\mathcal{S} = \{x \mid x_i \in \{0,1\}^n\}$, 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.

For example, consider the least-squares *regressor selection* or *feature selection* problem

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

which is to find the best fit to b from no more than k columns of A. For this problem, ADMM takes the form above, where the x-update involves only linear operations and the

z-update involves keeping the k largest magnitude elements of $x^{k+1}+u^k$; note that this is just like ADMM for the lasso, except that soft thresholding is replaced with 'hard' thresholding. However, there is no reason to believe that this converges to a global solution; indeed, it can converge to different points depending on the choice of ρ or the starting point z^0, as is usually the case in nonconvex optimization.

8.2 Bi-affine constraints

This section considers another nonconvex problem that admits exact ADMM updates:

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

where f and g are convex, and $F:\mathbf{R}^n\times\mathbf{R}^m\to\mathbf{R}^p$ is *bi-affine*, *i.e.*, affine in x for each fixed z, and affine in z for each fixed x. When F is jointly affine in x and z, this reduces to the standard ADMM problem form (9). For this problem ADMM has the form

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

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

For example, consider the problem of *nonnegative matrix factorization* [LS01]:

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

with variables $V\in\mathbf{R}^{p\times r}$, $W\in\mathbf{R}^{r\times q}$, and data $C\in\mathbf{R}^{p\times q}$. In ADMM form with bi-affine constraints, this can be written as

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

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

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

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 step splits in the columns of W, so can be performed by solving parallel quadratic programs to find each column.

9 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 how this maps onto a variety of software frameworks.

9.1 Abstract implementation

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

$$\begin{array}{rcl} y_i & := & y_i + \rho(x_i - z) \\ x_i & := & \operatorname{argmin}\left(f_i(x_i) + y_i^T(x_i - z) + (\rho/2)\|x_i - z\|_2^2\right) \\ z & := & \mathbf{prox}_{g,N\rho}\left(\overline{x} + (1/\rho)\overline{y}\right). \end{array}$$

Here, the iteration indices are omitted because in an actual implementation, we can simply overwrite previous values of these variables. Note that the y-update must be done before the x-update in order to match (28-30). If $g = 0$, then the z-update simply involves computing $\overline{x}$, and the y_i are not part of the aggregation, as discussed in §5.3.

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 y_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* [Tsi84, XB04]. 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 particular software framework used.

From the local perspective, each node i receives z, updates y_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 y_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, 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.

9.2 MPI

Message Passing Interface (MPI) [For09] is a language-independent message-passing specification used for parallel algorithms, and is the most widely used model for high-performance parallel computing today. Implementations of MPI are available in a wide variety of languages, including C, C++, and Python. There are different ways of implementing consensus ADMM in MPI, but perhaps the simplest is below; MPI operations are italicized.

Algorithm 1 *Global consensus ADMM in MPI.*

initialize a central collector process and N local processes, along with x_i, y_i, z.

repeat

if this is the central collector process

1. *Receive* (x_i, y_i, r_i), for $i = 1, \ldots, N$.
2. Let $z^{\text{prev}} := z$ and update $z := \mathbf{prox}_{g,N\rho}(\overline{x} + (1/\rho)\overline{y})$.
3. If $\rho\sqrt{N}\|z - z^{\text{prev}}\|_2 \leq \epsilon^{\text{conv}}$ and $(\sum_{i=1}^N \|r_i\|_2^2)^{1/2} \leq \epsilon^{\text{feas}}$, set CONVERGED.
4. *Send* $(z, \text{CONVERGED})$ to process i, for $i = 1, \ldots, N$.

else if this is local process i

1. Update $y_i := y_i + \rho(x_i - z)$.
2. Update $x_i := \text{argmin}_x \left(f_i(x) + y_i^T(x - z) + (\rho/2)\|x - z\|_2^2\right)$.
3. Compute $r_i := x_i - z$.
4. *Send* (x_i, y_i, r_i) to central collector.
5. *Receive* $(z, \text{CONVERGED})$ from central collector.

until CONVERGED.

A few comments on this pseudocode are needed. First, MPI typically uses a *single program, multiple data* (SPMD) model, meaning that the same program source code runs on each

separate machine, and conditional branches are used to execute different code on different machines (*i.e.*, a process can identify whether it is the central collector or a specific subsystem i). Second, the *send* and *receive* calls are blocking, and they unblock when a corresponding *receive* or *send* is executed, respectively. This provides the necessary synchronization between the subsystems and the collector. Third, the central collector receives (x_i, y_i, r_i) one at a time, as soon as they are sent by each given local process, regardless of the order in which the processes finish. There is also no need to store more than one (x_i, y_i, r_i) at any given time, since the quantities $\overline{x} + (1/\rho)\overline{y}$ and $\sum_{i=1}^{N} \|r_i\|_2^2$ can be computed in a streaming fashion, and the order in which the messages are received is clearly irrelevant.

In an extremely large scale implementation with many subsystems and potentially very high-dimensional variables, a more sophisticated message-passing scheme is necessary to avoid using excessive amounts of network bandwidth for the messages themselves. In this setting, a standard approach is to use a hierarchical topology, so each subsystem sends its messages to a 'regional collector', and the regional collectors aggregate and pass these messages on to the central collector. In addition, in-memory compression algorithms can be used to reduce the sizes of the messages themselves. We omit a further discussion since these are standard issues for large-scale MPI implementations and not specific to ADMM.

9.3 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 [Val90] 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 performing local y_i- and x_i-updates. The communication step would broadcast the new x_i and y_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 [GL05], GraphLab [LGK$^+$10], and Pregel [MAB$^+$10], among others. Since all three follow the general outline above, we refer the reader to the individual papers for details.

9.4 MapReduce

MapReduce [DG08] 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 [Whi10] for general background and motivation, and [Mur10] for some comments on best practices in setting up MapReduce tasks.

Hadoop is a software framework that contains a number of components supporting large-scale, fault-tolerant distributed computing applications, and is used very widely in industry for very large-scale distributed computing. The three components of primary relevance here are HDFS, a distributed file system based on Google's GFS [GGL03]; HBase, a distributed database based on Google's BigTable [CDG+08]; and, of course, the MapReduce engine itself. Because ADMM, an iterative algorithm, does not naturally fit into the batch-processing MapReduce paradigm, it is necessary to be aware of certain infrastructure details in addition to understanding MapReduce in the abstract.

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 should process a single block of data at a time, since a block is guaranteed to reside wholly on one machine; otherwise, one may unnecessarily access data across different machines to compute a result. In general, it is best to store a small number of large files and process them a block at a time.

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 framework then collects all the values $v'_1, \ldots, v'_r$ that correspond to the same output key k'_i (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.

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.

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. The major difficulty is that MapReduce tasks are not designed to be iterative and preserve state in the Mappers across iterations, which is necessary to implement ADMM.

We will describe a simple global consensus implementation to give the general flavor, and refer the reader to [LS10] for suggestions on how to improve this implementation and how best to extend to the general consensus case. Here, a local dataset $\mathcal{D}_i$ should be assumed to be a chunk of data comprising a single HDFS block, so neither an entire file nor a single training example. 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. Below, we use HBase, a distributed database built on top of HDFS that provides fast random read-write access.

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

function map(key i, dataset $\mathcal{D}_i$)

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

function reduce(key CENTRAL, records $(x_1, y_1), \ldots, (x_N, y_N)$)

1. Update $\hat{z} := \sum_{i=1}^N x_i + (1/\rho)y_i$.
2. *Emit* (key j, record $(x_j, y_j, \hat{z})$) to HBase for $j = 1, \ldots, N$.

Here, we have the Reducer compute $\hat{z} = \sum_{i=1}^N x_i + (1/\rho)y_i$ rather than z or $\tilde{z}$ because summation is associative while averaging is not. We assume N is always known. We have N

Mappers, one for each subsystem, and each Mapper updates y_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.)

It is worth briefly commenting on how to store the variables in HBase. HBase, like BigTable, is 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 [CDG+08, Whi10]. Also, note that we have discussed HBase since it is part of Hadoop, but there are other distributed key-value stores now available that may be more suitable for particular applications.

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 similar in concept 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 [DG08, Whi10, LS10] for further details.

10 Numerical examples

In this section, we report numerical results for a few 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 form ADMM to solve a distributed problem. The implementations of ADMM are written to be as simple as possible, with no optimization, customization, or tuning. All times are reported for a 3 GHz Intel Core 2 Duo processor.

10.1 Small dense lasso

We consider a small, dense instance of the lasso problem described in §6.4, where the $m \times n$ feature matrix A has $m = 1500$ examples and $n = 5000$ features. Each entry of A was sampled independently from a standard normal distribution, and the columns were then normalized to have unit norm. To generate the labels, we first generated a 'true' value $x^{\text{true}} \in \mathbf{R}^n$, with 100 nonzero entries, each sampled independently from a standard normal distribution. The labels b were then computed as $b = Ax^{\text{true}} + v$, where $v \sim \mathcal{N}(0, 10^{-3}I)$. This corresponds to a signal-to-noise ratio $\|Ax^{\text{true}}\|_2^2/\|v\|_2^2$ around 60.

The ADMM algorithm parameters were $\rho = 1$, and stopping criterion tolerances $\epsilon^{\text{abs}} = 10^{-4}$ and $\epsilon^{\text{rel}} = 10^{-2}$. The variables u^0 and z^0 were initialized to be zero.

We first solve a single instance of the problem, with a fixed regularization parameter λ, and then discuss computing the whole regularization path, in which we solve the problem for a wide range of regularization parameters.

Single problem. The parameter λ was chosen as $\lambda = 0.1\lambda_{\max}$, 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 2 shows the primal and dual residual norms versus 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 triggered after 15 iterations, but we ran ADMM for 35 iterations to show the continued progress. Figure 3 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` [KKL+07].)

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.

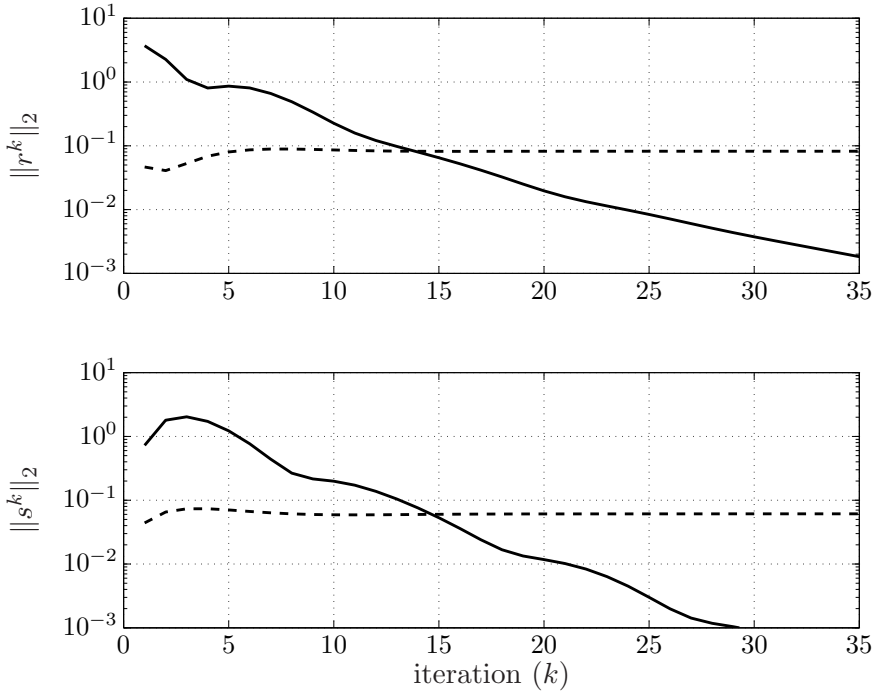

Figure 2: Norms of primal residual (top) and dual residual (bottom) versus iteration. The dashed lines show ϵ^{pri} (top) and ϵ^{dual} (bottom).

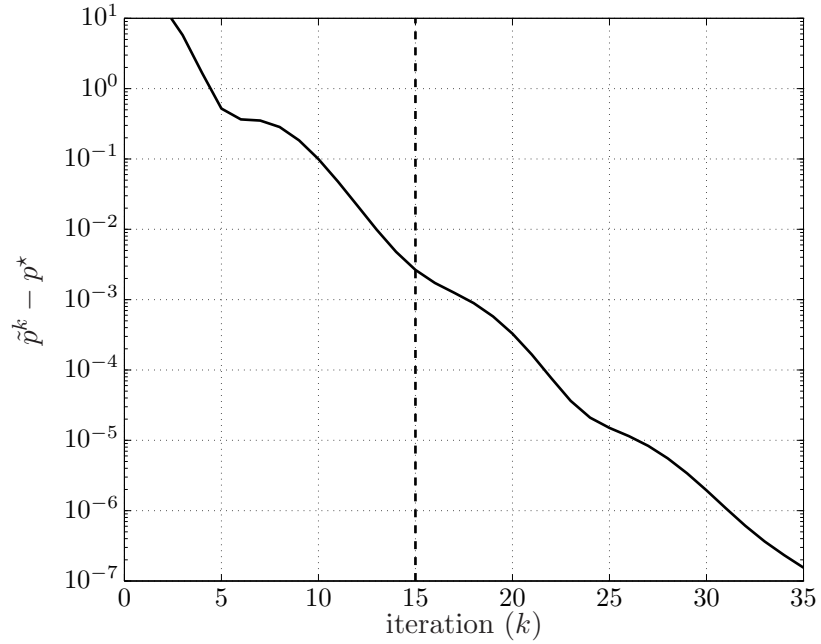

Figure 3: Objective suboptimality versus iterations. The stopping criterion is satisfied at iteration 15, indicated by the vertical dashed line.

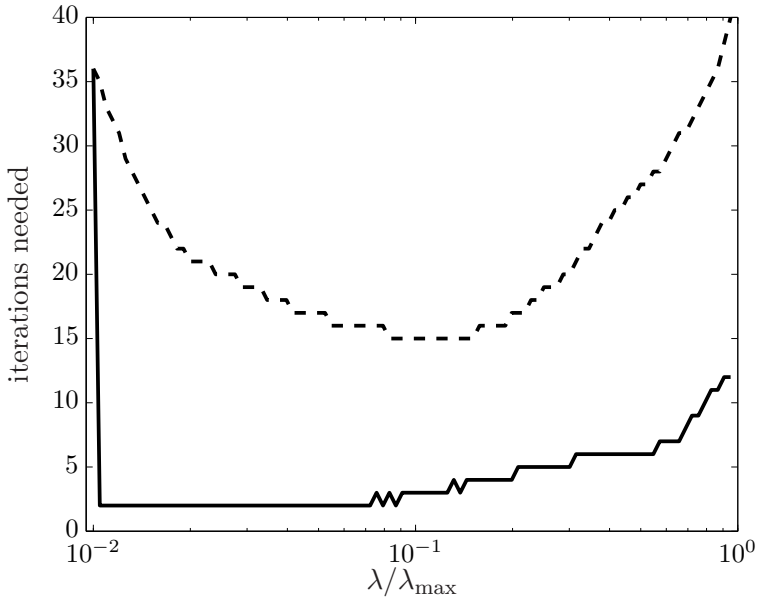

Figure 4: Iterations needed versus λ for warm start (solid line) and cold start (dashed line).

In our basic implementation, the factorization step costs about 1 second and the x-update costs around 30 ms. (This gives a factor/solve ratio of only 35, 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.

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.

Regularization path. We used 100 values of λ spaced logarithmically from $0.01\lambda_{\max}$ (in which case $x^\star$ has around 800 nonzeroes), to $0.95\lambda_{\max}$ (in which case $x^\star$ 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 and significantly reduces the number of iterations required to solve each lasso problem after the first one.

Figure 4 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 1.

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 1: Summary of timings for dense lasso example.

10.2 Distributed ℓ_1 regularized logistic regression

In this example we use consensus ADMM to fit an ℓ_1 regularized logistic regression model. Following §7, the problem is

$$\text{minimize} \quad \sum_{i=1}^{m} \log\left(1 + \exp(-b_i(a_i^T w + v))\right) + \lambda \|w\|_1, \tag{45}$$

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 choose 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

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

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

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 $A \in \mathbf{R}^{m \times n}$ have rows a_i^T. If θ^{neg} is the fraction of examples with $b_i = -1$ and θ^{pos} is the fraction with $b_i = 1$, then 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 [KKB07, §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 (40) in §7.2 with local variables $x_i = (v_i, w_i)$ and consensus variable $z = (v, w)$. As in the lasso example, we used

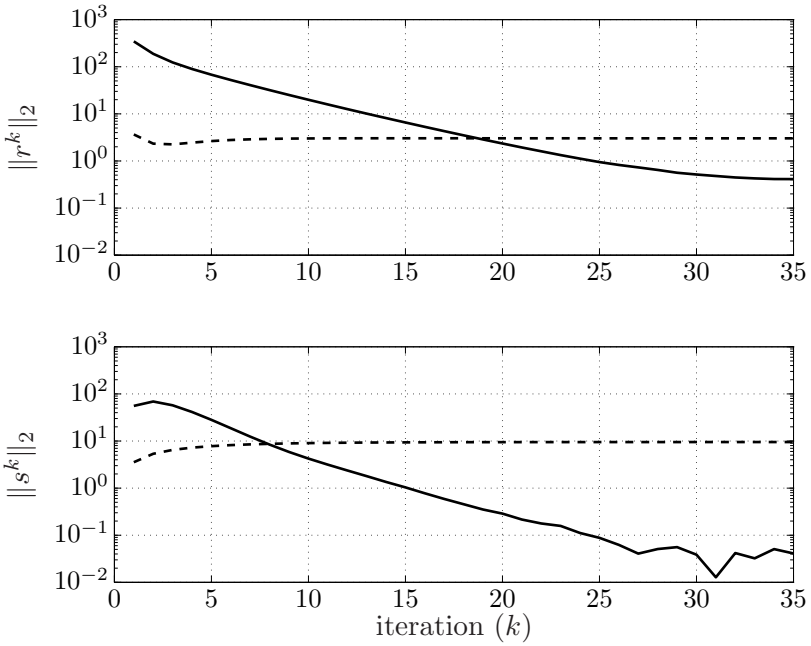

Figure 5: Progress of primal and dual residual norm. The dashed lines show ϵ^{pri} (top) and ϵ^{dual} (bottom).

$\epsilon^{\text{abs}} = 10^{-4}$ and $\epsilon^{\text{rel}} = 10^{-2}$ as tolerances and used the initialization $u_i^0 = 0, z^0 = 0$. We also use a fixed parameter of $\rho = 1$ for the iterations.

In this example, carrying out the x_i-update requires solving a smooth nonlinear optimization problem. To illustrate the use of an iterative method for performing these updates, we used the limited memory BFGS (L-BFGS) method [LN89, BLN95]. We used Nocedal's original 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). As suggested in §4.2, we warm started the x_i-updates using the values of x_i from the previous iteration.

To carry out the simulations for this example, we used a serial implementation that performs the x_i-updates sequentially. (In an actual 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 5 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 those of z up to the third digit.

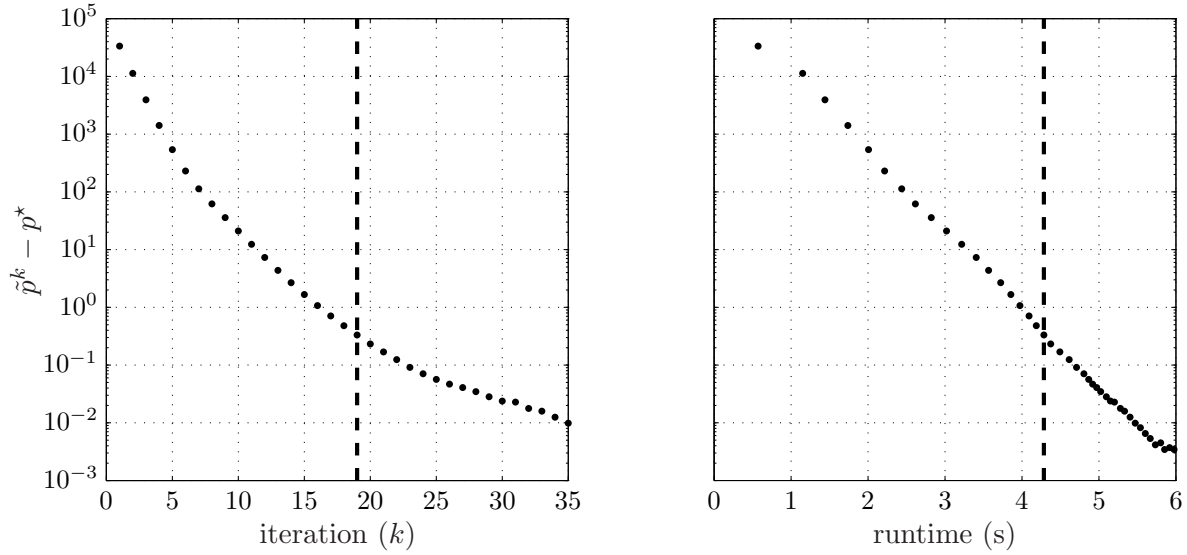

Figure 6: *Left.* Progress of distributed ADMM ℓ_1 regularized logistic regression versus iteration. *Right.* Progress versus elapsed time. The stopping criterion is satisfied at iteration 19, indicated by the vertical dashed line.

Figure 6 shows the suboptimality $\tilde{p}^k - p^\star$ for the consensus variable, where

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

and the optimal value $p^\star = 0.3302 \times 10^6$ was verified using `l1_logreg` [KKB07]. 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 take 2 seconds, while the last 4 iterations before the stopping criterion is satisfied take less than 0.5 seconds. This is because as the iterates approach consensus, L-BFGS requires fewer iterations due to warm starting.

11 Conclusions

In this paper, 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 a useful general purpose 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. What is surprising is that in some cases, a simple algorithm derived from basic ADMM will 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. Moreover, ADMM has the benefit of being extremely simple to implement, and it maps onto several standard 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.

Acknowledgements

We are very grateful to Rob Tibshirani and Trevor Hastie for encouraging us to write this paper. Thanks also to Dimitri Bertsekas, Danny Bickson, Tom Goldstein, Dimitri Gorinevsky, Daphne Koller, Vicente Malave, Stephen Oakley, and Alex Teichman for helpful comments and discussions.

A Convergence proof

The basic convergence result given in §3.2 can be found in several references, such as [Gab83, EB92]. 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

$$V^{k+1} \leq V^k - \|r^{k+1}\|_2^2 - \|B(z^{k+1} - z^k)\|_2^2. \tag{46}$$

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

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

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 (19), which requires the primal and dual residuals to be small, will eventually hold.)

The second key inequality is

$$p^{k+1} - p^\star \leq -(y^{k+1})^T r^{k+1} - \rho(B(z^{k+1} - z^k))^T \left(-r^{k+1} + B(z^{k+1} - z^\star)\right), \tag{47}$$

and the third inequality is

$$p^\star - p^{k+1} \leq y^{\star T} r^{k+1}. \tag{48}$$

The righthand side in (47) 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 (48) 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 (20) mentioned in our discussion of stopping criterion from the inequality (47). We simply observe that $-r^{k+1} + B(z^{k+1} - z^k) = -A(x^{k+1} - x^\star)$; substituting this into (47) yields (20),

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

Proof of inequality (48). 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 (48).

Proof of inequality (47). 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.*, [Roc70, §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

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

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 (47).

Proof of inequality (46). Adding (47) and (48), regrouping terms, and multiplying through by 2 gives

$$2(y^{k+1} - y^\star)^T r^{k+1} - 2\rho(B(z^{k+1} - z^k))^T r^{k+1} + 2\rho(B(z^{k+1} - z^k))^T (B(z^{k+1} - z^\star)) \leq 0. \quad (49)$$

The result (46) 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

$$(1/\rho)\left(\|y^{k+1} - y^\star\|_2^2 - \|y^k - y^\star\|_2^2\right) + \rho\|r^{k+1}\|_2^2. \tag{50}$$

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 (50). Substituting $z^{k+1} - z^\star = (z^{k+1} - z^k) + (z^k - z^\star)$ in the last term gives

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

and substituting $z^{k+1} - z^k = (z^{k+1} - z^\star) - (z^k - z^\star)$ in the last two terms, we get

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

Together with the previous step, this implies that (49) can be written as

$$V^k - V^{k+1} \geq \rho\|r^{k+1} - B(z^{k+1} - z^k)\|_2^2. \tag{51}$$

To show (46), 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 (51) 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 $\rho > 0$ by assumption.

References

[AHU58] K. J. Arrow, L. Hurwicz, and H. Uzawa. *Studies in Linear and Nonlinear Programming*. Stanford University Press: Stanford, 1958.

[AS58] K. J. Arrow and R. M. Solow. Gradient methods for constrained maxima, with weakened assumptions. In K. J. Arrow, L. Hurwicz, and H. Uzawa, editors, *Studies in Linear and Nonlinear Programming*. Stanford University Press: Stanford, 1958.

[BB94] H. H. Bauschke and J. M. Borwein. Dykstra's alternating projection algorithm for two sets. *Journal of Approximation Theory*, 79(3):418–443, 1994.

[BB96] H. H. Bauschke and J. M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.

[BBC09] 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.

[BDE09] 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*, 51(1):34–81, 2009.

[BDF10] J. M. Bioucas-Dias and M. A. T. Figueiredo. Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing. *arXiv:1002.4527*, 2010.

[Ben62] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[Ber82] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[BGd08] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learning Research*, 9:485–516, 2008.

[BJM06] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[BLN95] R. H. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[BLT76] 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*, pages 133–257, 1976.

[Bre65] L. M. Bregman. Finding the common point of convex sets by the method of successive projections. *Proceedings of the USSR Academy of Sciences*, 162(3):487–490, 1965.

[Bre67] 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*, 7(3):200–217, 1967.

[Bre73] H. Brezis. *Opérateurs maximaux monotones et semi-groupes de contractions dans les espaces de Hilbert*. North-Holland: Amsterdam, 1973.

[BT89] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[BT09] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. Available at `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`.

[CDG+08] 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*, 26(2):1–26, 2008.

[CDS01] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[CG59] W. Cheney and A. A. Goldstein. Proximity maps for convex sets. *Proceedings of the American Mathematical Society*, 10(3):448–450, 1959.

[CM73] J. F. Claerbout and F. Muir. Robust modeling with erratic data. *Geophysics*, 38:826, 1973.

[Com96] P. L. Combettes. The convex feasibility problem in image recovery. *Advances in Imaging and Electron Physics*, 95:155–270, 1996.

[CP07] 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*, 1(4):564–574, 2007.

[CP09] P. L. Combettes and J. C. Pesquet. Proximal splitting methods in signal processing. *arXiv:0912.3522*, 2009.

[CRT06] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489, 2006.

[CT94] G. Chen and M. Teboulle. A proximal-based decomposition method for convex minimization problems. *Mathematical Programming*, 64:81–101, 1994.

[CT06] E. J. Candès and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[CW06] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200, 2006.

[CZ92] Y. Censor and S. A. Zenios. Proximal minimization algorithm with D-functions. *Journal of Optimization Theory and Applications*, 73(3):451–464, 1992.

[CZ97] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, 1997.

[Dan63] G. B. Dantzig. *Linear programming and extensions*. RAND Corporation, 1963.

[DDM04] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57:1413–1457, 2004.

[Dem72] A. P. Dempster. Covariance selection. *Biometrics*, 28(1):157–175, 1972.

[Dem97] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM: Philadelphia, PA, 1997.

[DG08] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DGK08] J. Duchi, S. Gould, and D. Koller. Projected subgradient methods for learning sparse Gaussians. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2008.

[DMM09] D. L. Donoho, A. Maleki, and A. Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914, 2009.

[Don95] D. L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41:613–627, 1995.

[Don06] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[DR56] 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*, 82:421–439, 1956.

[DT06] D. L. Donoho and Y. Tsaig. Fast solution of ℓ_1-norm minimization problems when the solution may be sparse. Technical report, Stanford University, 2006.

[DW60] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[Dyk83] R. L. Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983.

[EB90] J. Eckstein and D. P. Bertsekas. An alternating direction method for linear programming. Technical report, MIT, 1990.

[EB92] J. Eckstein and D. P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[Eck89] J. Eckstein. *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, MIT, 1989.

[Eck93] J. Eckstein. Nonlinear proximal point algorithms using Bregman functions, with applications to convex programming. *Mathematics of Operations Research*, pages 202–226, 1993.

[Eck94a] J. Eckstein. Parallel alternating direction multiplier decomposition of convex programs. *Journal of Optimization Theory and Applications*, 80(1):39–62, 1994.

[Eck94b] J. Eckstein. Some saddle-function splitting methods for convex programming. *Optimization Methods and Software*, 4(1):75–83, 1994.

[Eck03] J. Eckstein. A practical general approximation criterion for methods of multipliers based on Bregman distances. *Mathematical Programming*, 96(1):61–86, 2003.

[EF93] J. Eckstein and M. Fukushima. Some reformulations and applications of the alternating direction method of multipliers. *Large Scale Optimization: State of the Art*, pages 119–138, 1993.

[EF98] 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*, 10(2):218–235, 1998.

[ES08] J. Eckstein and B. F. Svaiter. A family of projective splitting methods for the sum of two maximal monotone operators. *Mathematical Programming*, 111(1-2):173, 2008.

[ES09] J. Eckstein and B. F. Svaiter. General projective splitting methods for sums of maximal monotone operators. *SIAM Journal on Control and Optimization*, 48:787–811, 2009.

[Eve63] H. Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

[FBD10] M. A. T. Figueiredo and J. M. Bioucas-Dias. Restoration of Poissonian images using alternating direction optimization. *arXiv:1001.2244*, 2010.

[FCG10] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 11:1663–1707, 2010.

[FG83a] M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[FG83b] M. Fortin and R. Glowinski. On decomposition-coordination methods using an augmented Lagrangian. In M. Fortin and R. Glowinski, editors, *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[FHT08] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432, 2008.

[FM90] 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.

[FNW07] 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*, 1(4):586–597, 2007.

[For09] MPI Forum. *MPI: A Message-Passing Interface Standard, version 2.2*. High-Performance Computing Center: Stuttgart, 2009.

[FS95] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

[FS09] M. J. Fadili and J. L. Starck. Monotone operator splitting for optimization problems in sparse recovery. *IEEE ICIP*, 2009.

[Fuk92] M. Fukushima. Application of the alternating direction method of multipliers to separable convex programming problems. *Computational Optimization and Applications*, 1:93–111, 1992.

[Gab83] D. Gabay. Applications of the method of multipliers to variational inequalities. In M. Fortin and R. Glowinski, editors, *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[Geo72] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[GGL03] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[GL05] D. Gregor and A. Lumsdaine. The Parallel BGL: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing*, 2005.

[GM75] 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*, 9:41–76, 1975.

[GM76] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Computers and Mathematics with Applications*, 2:17–40, 1976.

[GO09] T. Goldstein and S. Osher. The split Bregman method for ℓ_1 regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[GT79] E. G. Gol'shtein and N. V. Tret'yakov. Modified Lagrangians in convex programming and their generalizations. *Point-to-Set Maps and Mathematical Programming*, pages 86–97, 1979.

[GT87] R. Glowinski and P. Le Tallec. Augmented Lagrangian methods for the solution of variational problems. Technical Report 2965, University of Wisconsin-Madison, 1987.

[GvL96] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[Hes69a] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:302–320, 1969.

[Hes69b] M. R. Hestenes. Multiplier and gradient methods. In L. A. Zadeh, L. W. Neustadt, and A. V. Balakrishnan, editors, *Computing Methods in Optimization Problems*. Academic Press, 1969.

[HNP09] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2), 2009.

[HT90] T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall, 1990.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, second edition, 2009.

[Hub64] P. J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[HYW00] 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*, 106(2):337–356, 2000.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[KKB07] 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*, 1(8):1519–1555, 2007.

[KKL+07] 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*, 1(4):606–617, 2007.

[KM98] S. A. Kontogiorgis and R. R. Meyer. A variable-penalty alternating directions method for convex optimization. *Mathematical Programming*, 83:29–53, 1998.

[Kon94] S. A. Kontogiorgis. *Alternating directions methods for the parallel solution of large-scale block-structured optimization problems*. PhD thesis, University of Wisconsin-Madison, 1994.

[Las70] L. S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, 1970.

[LGK+10] 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.

[LM79] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16:964–979, 1979.

[LN89] D. C. Liu and J. Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming B*, 45(3):503–528, 1989.

[LS87] J. Lawrence and J. E. Spingarn. On fixed points of non-expansive piecewise isometric mappings. *Proceedings of the London Mathematical Society*, 3(3):605, 1987.

[LS01] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13, 2001.

[LS10] 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*, pages 78–85, 2010.

[Lu09] Z. Lu. Smooth optimization approach for sparse covariance selection. *SIAM Journal on Optimization*, 19(4):1807–1827, 2009.

[Lue73] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley: Reading, MA, 1973.

[MAB+10] 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*, pages 135–146, 2010.

[MARS07] 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.

[MB06] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *Annals of Statistics*, 34(3):1436–1462, 2006.

[MCIL71] 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*, 8:115–130, 1971.

[MCL71] 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*, 8:131–153, 1971.

[MMLC72] 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*, 10:1–33, 1972.

[MN91] P. J. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, 1991.

[Mor62] J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace Hilbertien. *Reports of the Paris Academy of Sciences, Series A*, 255:2897–2899, 1962.

[Mur10] A. C. Murthy. Apache Hadoop: Best practices and anti-patterns. `http://developer.yahoo.com/blogs/hadoop/posts/2010/08/apache_hadoop_best_practices_a`, 2010.

[Nes83] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[Nes07] Y. Nesterov. Gradient methods for minimizing composite objective function. *CORE Discussion Paper, Catholic University of Louvain*, 76:2007, 2007.

[NO09] A. Nedić and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[NO10] A. Nedić and A. Ozdaglar. Cooperative distributed multi-agent optimization. In D. P. Palomar and Y. C. Eldar, editors, *Convex Optimization in Signal Processing and Communications*. Cambridge University Press, 2010.

[NWY09] M. Ng, P. Weiss, and X. Yuang. Solving constrained total-variation image restoration and reconstruction problems via alternating direction methods. *ICM Research Report*, 2009. Available at `http://www.optimization-online.org/DB_FILE/2009/10/2434.pdf`.

[Pow69] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*. Academic Press, 1969.

[PR55] 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*, 3:28–41, 1955.

[Roc70] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[Roc76a] R. T. Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1:97–116, 1976.

[Roc76b] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877, 1976.

[ROF92] L. Rudin, S. J. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[Rus89] A. Ruszczyński. An augmented Lagrangian decomposition method for block diagonal linear programming problems. *Operations Research Letters*, 8(5):287–294, 1989.

[Rus95] A. Ruszczyński. On convergence of an augmented Lagrangian decomposition method for sparse convex optimization. *Mathematics of Operations Research*, 20(3):634–656, 1995.

[RW91] R. T. Rockafellar and R. J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

[RW98] R. T. Rockafellar and R. J-B Wets. *Variational Analysis*. Springer-Verlag, 1998.

[Sho85] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag, 1985.

[Spi85] J. E. Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32:199–223, 1985.

[SS02] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[ST10] G. Steidl and T. Teuber. Removing multiplicative noise by Douglas-Rachford splitting methods. *Journal of Mathematical Imaging and Vision*, 36(2):168–184, 2010.

[TBA86] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[Tse91] P. Tseng. Applications of a splitting algorithm to decomposition in convex programming and variational inequalities. *SIAM Journal on Control and Optimization*, 29(1):119–138, 1991.

[Tse97] P. Tseng. Alternating projection-proximal methods for convex programming and variational inequalities. *SIAM Journal on Optimization*, 7:951–965, 1997.

[Tse00] P. Tseng. A modified forward-backward splitting method for maximal monotone mappings. *SIAM Journal on Control and Optimization*, 38:431, 2000.

[Tsi84] J. N. Tsitsiklis. *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, 1984.

[TVSL10] 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*, 11:311–365, 2010.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):111, 1990.

[Vap00] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 2000.

[vN50] 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.

[Whi10] T. White. *Hadoop: The Definitive Guide*. O'Reilly Press, second edition, 2010.

[WL01] 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*, 109(2):415–429, 2001.

[Woo09] J. M. Wooldridge. *Introductory Econometrics: A Modern Approach*. South Western College Publications, fourth edition, 2009.

[XB04] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[YGZ+10] 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.

[YL06] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[YOGD08] 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*, 1(1):143–168, 2008.

[Yua09] X. M. Yuan. Alternating direction methods for sparse covariance selection. *Preprint*, 2009. Available at `http://www.optimization-online.org/DB_FILE/2009/09/2390.pdf`.

[YY10] J. Yang and X. Yuan. An inexact alternating direction method for trace norm regularized least squares problem, 2010. Available at `http://www.optimization-online.org`.

[YZ09] J. Yang and Y. Zhang. Alternating direction algorithms for ℓ_1-problems in compressive sensing. *Preprint*, 2009.

[ZH05] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67:301–320, 2005.

[Zha04] T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 32(1):56–85, 2004.