From BOYD ET AL ; see link on course web-page.

2

Precursors

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

2.1 Dual Ascent

Consider the equality-constrained convex optimization problem

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

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

The Lagrangian for problem (2.1) is

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

and the dual function is

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

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

$$= -\underbrace{\sup_x \left(-f(x) - (A^T y)^T x\right)}_{-f^*(-A^T y)} - b^T y$$

problem is

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

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

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

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

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

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

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

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

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

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

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

2.2 Dual Decomposition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3 Augmented Lagrangians and the Method of Multipliers

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

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

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

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

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

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

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

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

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

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

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

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

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

i.e. $A^T y_{k+1} + \nabla f(x_{k+1}) = 0$

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

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

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

3

Alternating Direction Method of Multipliers

3.1 Algorithm

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

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

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

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

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

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

ADMM consists of the iterations

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

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

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

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

The method of multipliers for (3.1) has the form

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

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

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

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

3.1.1 Scaled Form

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

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

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

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

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

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

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

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

the running sum of the residuals.

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

3.2 Convergence

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

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

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

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

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

is a closed nonempty convex set.

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

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

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

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

holds for all x, z, y.

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

3.2.1 Convergence

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

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

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

3.2.2 Convergence in Practice

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

3.3 Optimality Conditions and Stopping Criterion

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

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

and dual feasibility,

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

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

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

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

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

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

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

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

or equivalently,

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

This means that the quantity

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

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

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

3.3.1 Stopping Criteria

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

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

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

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

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

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

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

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

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

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

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

3.4 Extensions and Variations

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

3.4.1 Varying Penalty Parameter

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

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

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

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

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

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

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

3.4.2 More General Augmenting Terms

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

3.4.3 Over-relaxation

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

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

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

3.4.4 Inexact Minimization

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

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

3.4.5 Update Ordering

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

3.4.6 Related Algorithms

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

3.5 Notes and References

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

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

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

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

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

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

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

Eckstein (one of the authors of this paper) told me he doesn't like the proof of convergence in the appendix - which is based on work of Fortin + Glowinski - because it is not at all intuitive. He prefers Gabay's approach, as given in his (Eckstein's) recent (2015) paper in Pacific J. Optim. However, that requires a lot of convex analysis. The proof in the appendix is much more straightforward, even if not intuitive.

4

General Patterns

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

$$x^+ = \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|Ax - v\|_2^2\right), \qquad \text{(see (3.5))}$$

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

4.1 Proximity Operator

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

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

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

(or proximal)

often denoted $\text{prox}_f(v)$ (when $\rho = 1$)

analysis,

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

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

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

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

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

4.2 Quadratic Objective Terms

Suppose f is given by the (convex) quadratic function

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

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

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

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

4.2.1 Direct Methods

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

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

4.2.2 Exploiting Sparsity

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

4.2.3 Caching Factorizations

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

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

4.2.4 Matrix Inversion Lemma

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

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

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

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

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

4.2.5 Quadratic Function Restricted to an Affine Set

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

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

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

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

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

4.3 Smooth Objective Terms

4.3.1 Iterative Solvers

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

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

4.3.2 Early Termination

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

4.3.3 Warm Start

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

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

4.3.4 Quadratic Objective Terms

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

4.4 Decomposition

4.4.1 Block Separability

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

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

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

4.4.2 Component Separability

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

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

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

4.4.3 Soft Thresholding

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

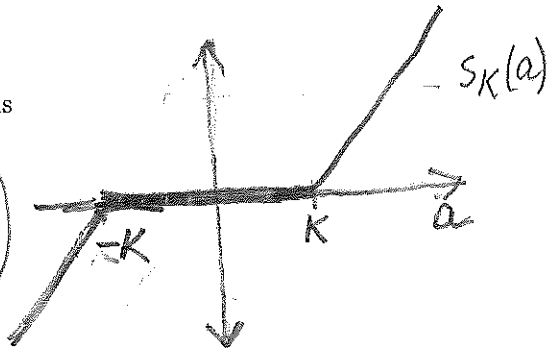
$$x_i^+ := \operatorname*{argmin}_{x_i} \left(\lambda|x_i| + (\rho/2)(x_i - v_i)^2\right).$$

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

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

where the *soft thresholding operator* S is defined as

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



or equivalently,

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

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

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

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

(separable wrt components of x)

Let $h^{(i)}(x) = \lambda|x| + \frac{\rho}{2}(x - v_i)^2$. Want $x_i^+ = \text{MINIMIZER of } h^{(i)}$.

Need $0 \in \partial h(x_i)$. We have $\partial h^{(i)}(0) = [-\lambda, \lambda] - \rho v_i$.

and $\partial h^{(i)}(x) = \lambda \operatorname{sgn}(x) + \rho x - \rho v_i$ for $x \neq 0$.

We see $0 \in \partial h^{(i)}(0)$ iff $|v_i| \le \lambda/\rho$ so $x_i^+ = 0$ if $|v_i| \le \frac{\lambda}{\rho}$.

Otherwise need $(\lambda/\rho)\operatorname{sgn}(x) + x - v_i = 0$

If $v_i > \lambda/\rho$, set $x = v_i - \lambda/\rho > 0$: satisfies

If $v_i < -\lambda/\rho$, set $x = v_i + \lambda/\rho < 0$: satisfies.

5

Constrained Convex Optimization

The generic constrained convex optimization problem is

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

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

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

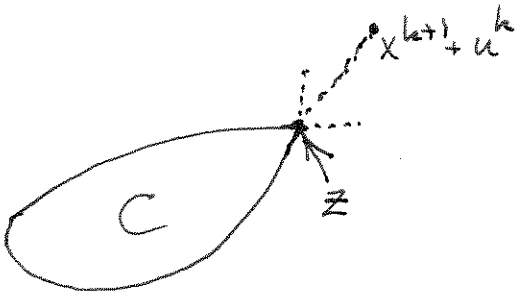
where g is the indicator function of $\mathcal{C}$.

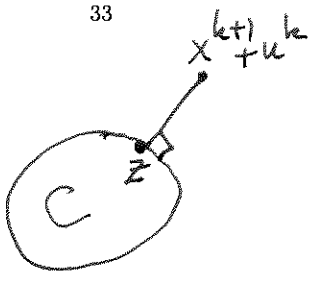
The augmented Lagrangian (using the scaled dual variable) is

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

so the scaled form of ADMM for this problem is

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





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

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

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

(see p. 15, p. 18)

In the following sections we give some more specific examples.

5.1 Convex Feasibility

5.1.1 Alternating Projections

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

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

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

In ADMM form, the problem can be written as

$A = I, B = -I, c = 0.$

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

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

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

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

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

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

5.1.2 Parallel Projections

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

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

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

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

and projection onto $\mathcal{D}$ is

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

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

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

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

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

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

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

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

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

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

5.2 Linear and Quadratic Programming

The standard form quadratic program (QP) is

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

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

We express it in ADMM form as

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

where

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

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

The scaled form of ADMM consists of the iterations

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

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

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

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

5.2.1 Linear and Quadratic Cone Programming

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

Least Absolute Shrinkage & Selection Operator

not always used to mean same thing

See HW6

6.4 Lasso

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

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

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

In ADMM form, the lasso problem can be written as

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

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

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

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

STOP & go to HERE conv. and.

6.4.1 Generalized Lasso

The lasso problem can be generalized to

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

(see (3.5))

$$x^{k+1} = \arg\min_x \frac{1}{2}\|Ax-b\|^2 + \frac{\rho}{2}\|x - z^k + u^k\|^2$$

Df: $0 = (A^TA)x - A^Tb + \rho x - \rho(z^k - u^k)$

(see (3.6))

$$z^{k+1} = \arg\min_z \lambda\|z\|_1 + \frac{\rho}{2}\|\underbrace{x^{k+1} - z + u^k}_{(x^{k+1}+u^k)-z}\|^2$$

A

Convergence Proof

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

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

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

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

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

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

along with simple algebra. The first inequality is

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

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

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

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

The second key inequality is

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

and the third inequality is

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

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

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

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

Proof of inequality (A.3)

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

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

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

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

which gives (A.3).

Proof of inequality (A.2)

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

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

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

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

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

This implies that x^{k+1} minimizes

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

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

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

and that

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

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

Proof of inequality (A.1) (omitted)

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

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

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

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

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

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

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

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

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

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

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

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

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

in the last term gives

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

and substituting

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

in the last two terms, we get

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

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

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

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

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

and

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

to get that

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

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