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

7

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

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

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. $A^T y_{k+1} \overset{?}{\neq} \nabla f(x_{k+1}) = 0$

~~$Ax^{k+1}y - b$~~

12   Precursors

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

[handwritten annotation: TOTALLY NEW USE OF g!]

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

$$\text{minimize} \quad f(x) + g(z)$$
$$\text{subject to} \quad Ax + Bz = c \tag{3.1}$$

[handwritten annotation: vars are $x \in \mathbb{R}^n, z \in \mathbb{R}^m$]

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

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

13

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

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

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

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.

Handwritten top:

$$\left\| r + \frac{1}{\rho} y \right\|^2 = \| r \|^2 + \frac{2}{\rho} r^T y + \frac{1}{\rho^2} \| y \|^2$$

$$\frac{\rho}{2} \left\| r + \frac{1}{\rho} y \right\|^2 = \frac{\rho}{2} \| r \|^2 + r^T y + \frac{1}{2\rho} \| y \|^2$$

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

Handwritten left: (LHS is 2nd/3rd terms of $L_\rho(x, z^k, y^k)$)

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

Handwritten right: $\|y\|^2 = \rho^2 \|u\|^2$

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

Handwritten: r (subst z^k for z)

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

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

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

Handwritten right: (dropping $\frac{\rho}{2}\|u\|^2$ which is indep of x)

Handwritten right: r (subst x^{k+1} for x)

Handwritten left:
$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$
$$\rho \cdot u^{k+1} = \rho u^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$

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

Handwritten: (primal)

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

Handwritten right: SAVE ON BLACKBOARD

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

$\exists x \ s.t. \ f(x) < \infty$

$\forall x, \ f(x) > -\infty.$

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   *NOTE*

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

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 ~~omit~~

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

$$f(x^k) + g(z^k) - p^\star \le -(y^k)^T r^k + (x^k - x^\star)^T s^k. \qquad (3.11)$$

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 $\|x^k - x^\star\|_2 \le d$, we have that

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

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.*,

$$\|r^k\|_2 \le \epsilon^{\text{pri}} \quad \text{and} \quad \|s^k\|_2 \le \epsilon^{\text{dual}}, \qquad (3.12)$$

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

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

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

Skip p. 20–24,

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

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

(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

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 C, then the x-update is

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

where Π_C denotes projection (in the Euclidean norm) onto 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.

$f(x) = \begin{cases} 0 & \text{if } x \in C \\ \infty & \text{if } x \notin C \end{cases}$

see picture on p.33

4.2 Quadratic Objective Terms

Suppose f is given by the (convex) quadratic function

scalar, new use of r

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

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.

as x minimizes $\frac{1}{2}x^T P x + q^T x + \frac{\rho}{2}(Ax - v)^T(Ax - v) + \text{const}$

i.e. $\frac{1}{2}x^T(P + \rho A^T A)x + (q - \rho A^T v)^T x + \text{const}$.

now differentiate wrt x.

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* $F = F_1 F_2 \cdots F_k$ 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 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 $F = P + \rho A^T A$ and the right-hand side is $g = \rho A^T v - q$, 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 $F = P + \rho A^T A$ 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 $O(p^2 n)$ 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 $P + \rho A^T A$ 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.

Yet another g.

4.2.5 Quadratic Function Restricted to an Affine Set

Assuming $A = I$ here and $B = -I$

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

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

F [ ] ℓ rows, n

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.

$$x^+ = \arg\min \tfrac{1}{2} x^T P x + q^T x + \tfrac{\rho}{2} \| x - z^k + u^k) \| \equiv h(x)$$
$$\text{s.t.} \quad Fx = g.$$

KKT says $\nabla h(x^+) + F^T \nu = 0$ for some $\nu \in \mathbb{R}^\ell$ (from standard duality theory — see p. 243-244 in BV)

i.e. $P x^+ + q + \rho x^+ + \rho(-z^k + u^k) + F^T \nu = 0$ $\Big\} \equiv (\dagger)$
$$Fx = g$$

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.*, $A^T A$ 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 $A^T A$ 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

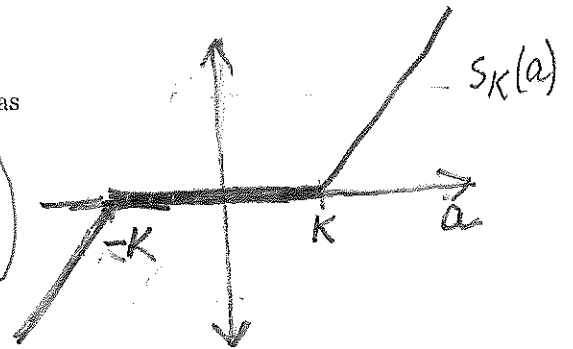$$x_i^+ := \operatorname*{argmin}_{x_i} \left(\lambda|x_i| + (\rho/2)(x_i - v_i)^2\right).$$

(separable w.r.t. components of x)

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

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

$S_K(a)$

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 $a \ne 0$). 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.

see proxabs.m

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

Need $0 \in \partial h^{(i)}(x_i)$. 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 $x \ne 0$.

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

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

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

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

$\begin{cases} g(z) = 0 & \text{if } z \in \mathcal{C} \\ g(z) = \infty & \text{if } z \notin \mathcal{C}. \end{cases}$

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

The augmented Lagrangian (using the scaled dual variable) is

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

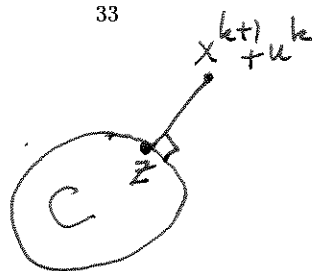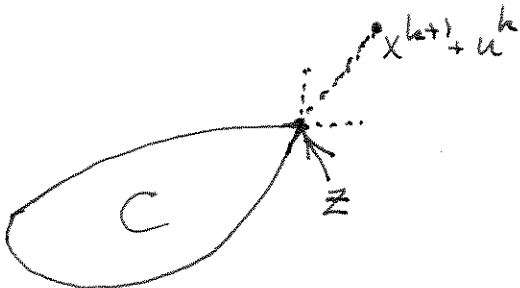so the scaled form of ADMM for this problem is

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

because need $g(z) = 0$, i.e. $z \in \mathcal{C}$, and then want to minimize $\| z - (x^{k+1} + u^k) \|$ subject to $z \in \mathcal{C}$

33

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

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

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

In the following sections we give some more specific examples.

5.1   Convex Feasibility

5.1.1   Alternating Projections

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

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

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

In ADMM form, the problem can be written as

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

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

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

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 $\mathrm{dist}(\mathcal{C}, \mathcal{D})$, the Euclidean distance between $\mathcal{C}$ and $\mathcal{D}$. If we terminate with $\|r^k\|_2 \leq \epsilon^{\mathrm{pri}}$, 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 $\epsilon^{\mathrm{pri}}/2$ 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

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

where $\bar{x} = (1/N) \sum_{i=1}^{N} x_i$ 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:

$$x_i^{k+1} := \Pi_{\mathcal{A}_i}(z^k - u_i^k)$$

$$z^{k+1} := \frac{1}{N} \sum_{i=1}^{N} (x_i^{k+1} + u_i^k)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}.$$

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

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

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

$$x_i^{k+1} := \Pi_{\mathcal{A}_i}(\bar{x}^k - u_i^k)$$
$$u_i^{k+1} := u_i^k + (x_i^{k+1} - \bar{x}^{k+1}).$$

This shows that u_i^k is the running sum of the the 'discrepancies' $x_i^k - \bar{x}^k$ (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

$$x^{k+1} := \underset{x}{\operatorname{argmin}} \left(f(x) + (\rho/2)\|x - z^k + u^k\|_2^2 \right) \quad \longleftarrow \text{IMPLICIT CONSTRAINT} \; Ax = b.$$
$$z^{k+1} := (x^{k+1} + u^k)_+ \quad \longleftarrow \text{PROJECTION ONTO } \mathbb{R}^n_+.$$
$$u^{k+1} := u^k + x^{k+1} - z^{k+1}.$$

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

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

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

5.2.1 Linear and Quadratic Cone Programming

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

6

ℓ_1-Norm Problems

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

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

actually, to the early 19th C, predating ℓ₂!

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

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

6.1 Least Absolute Deviations

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

In ADMM form, the problem can be written as

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

(skip)

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

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

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

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

6.1.1   Huber Fitting

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

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

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

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

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

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

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

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

6.2 Basis Pursuit

Basis pursuit is the equality-constrained ℓ_1 minimization problem

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

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

In ADMM form, basis pursuit can be written as

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

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

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

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

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

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

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

6.3   General ℓ_1 Regularized Loss Minimization

Consider the generic problem

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

where l is any convex loss function.

In ADMM form, this problem can be written as

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

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

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

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

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

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

6.4 [Lasso]

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

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

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

In ADMM form, the lasso problem can be written as

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

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

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

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

6.4.1 Generalized Lasso

The lasso problem can be generalized to

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

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

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

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

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

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

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

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

which yields the ADMM algorithm

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

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

6.4.2 Group Lasso

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

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

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

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

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

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

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

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

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

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

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

6.5 Sparse Inverse Covariance Selection

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

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

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

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

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

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

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

The ADMM algorithm for sparse inverse covariance selection is

$$X^{k+1} := \underset{X}{\text{argmin}} \left(\text{Tr}(SX) - \log\det X + (\rho/2)\|X - Z^k + U^k\|_F^2 \right)$$

$$Z^{k+1} := \underset{Z}{\text{argmin}} \left(\lambda\|Z\|_1 + (\rho/2)\|X^{k+1} - Z + U^k\|_F^2 \right)$$

$$U^{k+1} := U^k + X^{k+1} - Z^{k+1},$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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