Foundations and Trends® in Optimization
Vol. 1, No. 3 (2014) 127–239
© 2014 N. Parikh and S. Boyd
DOI: 10.1561/2400000003

Proximal Algorithms

Neal Parikh
Department of Computer Science
Stanford University
npparikh@cs.stanford.edu

Stephen Boyd
Department of Electrical Engineering
Stanford University
boyd@stanford.edu

Contents

Abstract

This monograph is about a class of optimization algorithms called *proximal algorithms*. Much like Newton's method is a standard tool for solving unconstrained smooth optimization problems of modest size, proximal algorithms can be viewed as an analogous tool for nonsmooth, constrained, large-scale, or distributed versions of these problems. They are very generally applicable, but are especially well-suited to problems of substantial recent interest involving large or high-dimensional datasets. Proximal methods sit at a higher level of abstraction than classical algorithms like Newton's method: the base operation is evaluating the *proximal operator* of a function, which itself involves solving a small convex optimization problem. These subproblems, which generalize the problem of projecting a point onto a convex set, often admit closed-form solutions or can be solved very quickly with standard or simple specialized methods. Here, we discuss the many different interpretations of proximal operators and algorithms, describe their connections to many other topics in optimization and applied mathematics, survey some popular algorithms, and provide a large number of examples of proximal operators that commonly arise in practice.

N. Parikh and S. Boyd. *Proximal Algorithms*. Foundations and Trends® in Optimization, vol. 1, no. 3, pp. 127–239, 2014.

DOI: 10.1561/2400000003.

1

Introduction

This monograph is about a class of algorithms, called *proximal algorithms*, for solving convex optimization problems. Much like Newton's method is a standard tool for solving unconstrained smooth minimization problems of modest size, proximal algorithms can be viewed as an analogous tool for nonsmooth, constrained, large-scale, or distributed versions of these problems. They are very generally applicable, but they turn out to be especially well-suited to problems of recent and widespread interest involving large or high-dimensional datasets.

Proximal methods sit at a higher level of abstraction than classical optimization algorithms like Newton's method. In the latter, the base operations are low-level, consisting of linear algebra operations and the computation of gradients and Hessians. In proximal algorithms, the base operation is evaluating the *proximal operator* of a function, which involves solving a small convex optimization problem. These subproblems can be solved with standard methods, but they often admit closed-form solutions or can be solved very quickly with simple specialized methods. We will also see that proximal operators and proximal algorithms have a number of interesting interpretations and are connected to many different topics in optimization and applied mathematics.

1.1 Definition

Let $f : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ be a closed proper convex function, which means that its *epigraph*

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

is a nonempty closed convex set. The *effective domain* of f is

$$\mathbf{dom}\, f = \{x \in \mathbf{R}^n \mid f(x) < +\infty\},$$

i.e., the set of points for which f takes on finite values.

The *proximal operator* $\mathbf{prox}_f : \mathbf{R}^n \to \mathbf{R}^n$ of f is defined by

$$\mathbf{prox}_f(v) = \operatorname*{argmin}_x \left(f(x) + (1/2)\|x - v\|_2^2 \right), \tag{1.1}$$

where $\|\cdot\|_2$ is the usual Euclidean norm. The function minimized on the righthand side is strongly convex and not everywhere infinite, so it has a unique minimizer for every $v \in \mathbf{R}^n$ (even when $\mathbf{dom}\, f \subsetneq \mathbf{R}^n$).

We will often encounter the proximal operator of the scaled function λf, where $\lambda > 0$, which can be expressed as

$$\mathbf{prox}_{\lambda f}(v) = \operatorname*{argmin}_x \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right). \tag{1.2}$$

This is also called the proximal operator of f with parameter λ. (To keep notation light, we write $(1/2\lambda)$ rather than $(1/(2\lambda))$.)

Throughout this monograph, when we refer to the proximal operator of a function, the function will be assumed to be closed proper convex, and it may take on the extended value $+\infty$.

1.2 Interpretations

Figure 1.1 depicts what a proximal operator does. The thin black lines are level curves of a convex function f; the thicker black line indicates the boundary of its domain. Evaluating $\mathbf{prox}_f$ at the blue points moves them to the corresponding red points. The three points in the domain of the function stay in the domain and move towards the minimum of the function, while the other two move to the boundary of the domain and towards the minimum of the function. The parameter λ controls

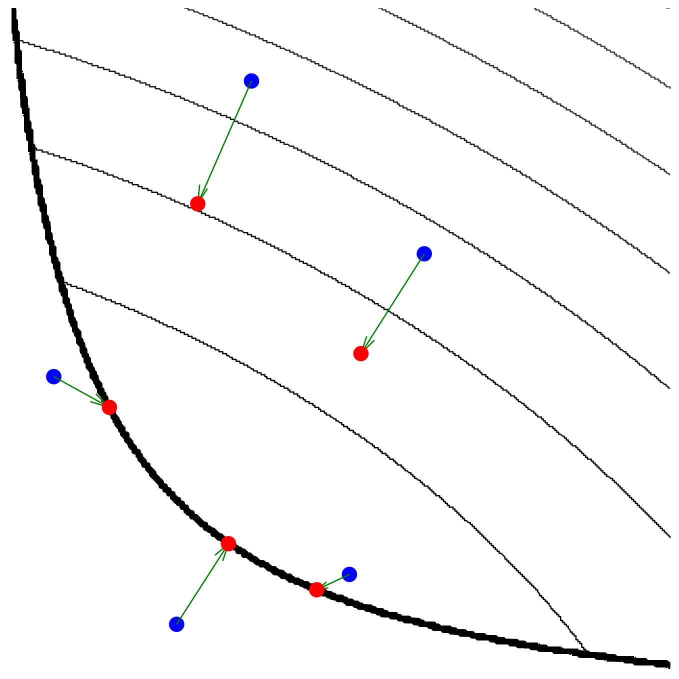

Figure 1.1: Evaluating a proximal operator at various points.

the extent to which the proximal operator maps points towards the minimum of f, with larger values of λ associated with mapped points near the minimum, and smaller values giving a smaller movement towards the minimum. It may be useful to keep this figure in mind when reading about the subsequent interpretations.

We now briefly describe some basic interpretations of (1.1) that we will revisit in more detail later. The definition indicates that $\mathbf{prox}_f(v)$ is a point that compromises between minimizing f and being near to v. For this reason, $\mathbf{prox}_f(v)$ is sometimes called a *proximal point* of v with respect to f. In $\mathbf{prox}_{\lambda f}$, the parameter λ can be interpreted as a relative weight or trade-off parameter between these terms.

When f is the *indicator function*

$$I_{\mathcal{C}}(x) = \begin{cases} 0 & x \in \mathcal{C} \\ +\infty & x \notin \mathcal{C}, \end{cases}$$

where $\mathcal{C}$ is a closed nonempty convex set, the proximal operator of f reduces to Euclidean projection onto $\mathcal{C}$, which we denote

$$\Pi_{\mathcal{C}}(v) = \operatorname*{argmin}_{x \in \mathcal{C}} \|x - v\|_2. \tag{1.3}$$

Proximal operators can thus be viewed as generalized projections, and this perspective suggests various properties that we expect proximal operators to obey.

The proximal operator of f can also be interpreted as a kind of gradient step for the function f. In particular, we have (under some assumptions described later) that

$$\mathbf{prox}_{\lambda f}(v) \approx v - \lambda \nabla f(v)$$

when λ is small and f is differentiable. This suggests a close connection between proximal operators and gradient methods, and also hints that the proximal operator may be useful in optimization. It also suggests that λ will play a role similar to a step size in a gradient method.

Finally, the fixed points of the proximal operator of f are precisely the minimizers of f (we will show this in §2.3). In other words, $\mathbf{prox}_{\lambda f}(x^\star) = x^\star$ if and only if $x^\star$ minimizes f. This implies a close connection between proximal operators and fixed point theory, and suggests that proximal algorithms can be interpreted as solving optimization problems by finding fixed points of appropriate operators.

1.3 Proximal algorithms

A *proximal algorithm* is an algorithm for solving a convex optimization problem that uses the proximal operators of the objective terms. For example, the *proximal minimization algorithm*, discussed in more detail in §4.1, minimizes a convex function f by repeatedly applying $\mathbf{prox}_f$ to some initial point x^0. The interpretations of $\mathbf{prox}_f$ above suggest several potential perspectives on this algorithm, such as an approximate gradient method or a fixed point iteration. In Chapters 4 and 5 we will encounter less trivial and far more useful proximal algorithms.

Proximal algorithms are most useful when all the relevant proximal operators can be evaluated sufficiently quickly. In Chapter 6, we discuss how to evaluate proximal operators and provide many examples.

There are many reasons to study proximal algorithms. First, they work under extremely general conditions, including cases where the functions are nonsmooth and extended real-valued (so they contain implicit constraints). Second, they can be fast, since there can be simple proximal operators for functions that are otherwise challenging to handle in an optimization problem. Third, they are amenable to distributed optimization, so they can be used to solve very large scale problems. Finally, they are often conceptually and mathematically simple, so they are easy to understand, derive, and implement for a particular problem. Indeed, many proximal algorithms can be interpreted as generalizations of other well-known and widely used algorithms, like the projected gradient method, so they are a natural addition to the basic optimization toolbox for anyone who uses convex optimization.

1.4 What this paper is about

We aim to provide a readable reference on proximal operators and proximal algorithms for a wide audience. There are several novel aspects.

First, we discuss a large number of different perspectives on proximal operators, some of which have not previously appeared in the literature, and many of which have not been collected in one place. These include interpretations based on projection operators, smoothing and regularization, resolvent operators, and differential equations. Second, we place strong emphasis on practical use, so we provide many examples of proximal operators that are efficient to evaluate. Third, we have a more detailed discussion of distributed optimization algorithms than most previous references on proximal operators.

To keep the treatment accessible, we have omitted a few more advanced topics, such as the connection to monotone operator theory.

We also include source code for all examples, as well as a library of implementations of proximal operators, at

`http://www.stanford.edu/~boyd/papers/prox_algs.html`

We provide links to other libraries of proximal operators, such as those by Becker et al. and Vaiter, in the documentation for our own library.

1.5 Related work

We emphasize that proximal operators are not new and that there have been other surveys written on various aspects of this topic over the years. Lemaire [123] surveys the literature on the proximal point algorithm up to 1989. Iusem [110] reviews the proximal point method and its connection to augmented Lagrangians. An excellent recent reference by Combettes and Pesquet [63] discusses proximal operators and proximal algorithms in the context of signal processing problems. The lecture notes for Vandenberghe's EE 236C course [196] covers proximal algorithms in detail. Finally, the recent monograph by Boyd et al. [33] is about a particular algorithm (ADMM), but also discusses connections to proximal operators. We will discuss more of the history of proximal operators in the sequel.

1.6 Outline

In Chapter 2, we give some basic properties of proximal operators. In Chapter 3, we discuss a variety of interpretations of proximal operators. Chapter 4 covers some core proximal algorithms for solving convex optimization problems. In Chapter 5, we discuss how to use these algorithms to solve problems in a parallel or distributed fashion. Chapter 6 presents a large number of examples of different projection and proximal operators that can be evaluated efficiently. In Chapter 7, we illustrate these ideas with some examples and applications.

2

Properties

We begin by discussing the main properties of proximal operators. These are used to, for example, establish convergence of a proximal algorithm or to derive a method for evaluating the proximal operator of a given function. All of these properties are well-known in the literature; see, *e.g.*, [63, 195, 10].

2.1 Separable sum

If f is separable across two variables, so $f(x, y) = \varphi(x) + \psi(y)$, then

$$\mathbf{prox}_f(v, w) = (\mathbf{prox}_\varphi(v), \mathbf{prox}_\psi(w)). \tag{2.1}$$

Thus, evaluating the proximal operator of a separable function reduces to evaluating the proximal operators for each of the separable parts, which can be done independently.

If f is fully separable, meaning that $f(x) = \sum_{i=1}^n f_i(x_i)$, then

$$(\mathbf{prox}_f(v))_i = \mathbf{prox}_{f_i}(v_i).$$

In other words, this case reduces to evaluating proximal operators of scalar functions. We will see in Chapter 5 that the separable sum property is the key to deriving parallel versions of proximal algorithms.

2.2 Basic operations

This section can be referred to as needed; these properties will not play a central role in the rest of the paper.

Postcomposition. If $f(x) = \alpha\varphi(x) + b$, with $\alpha > 0$, then

$$\mathbf{prox}_{\lambda f}(v) = \mathbf{prox}_{\alpha\lambda\varphi}(v).$$

Precomposition. If $f(x) = \varphi(\alpha x + b)$, with $\alpha \neq 0$, then

$$\mathbf{prox}_{\lambda f}(v) = \frac{1}{\alpha}\left(\mathbf{prox}_{\alpha^2\lambda\varphi}(\alpha v + b) - b\right). \tag{2.2}$$

If $f(x) = \varphi(Qx)$, where Q is orthogonal ($QQ^T = Q^TQ = I$), then

$$\mathbf{prox}_{\lambda f}(v) = Q^T\mathbf{prox}_{\lambda\varphi}(Qv).$$

There are other specialized results about evaluating $\mathbf{prox}_f$ via $\mathbf{prox}_\varphi$, where $f(x) = \varphi(Ax)$ for some matrix A. Several of these are useful in image and signal processing; see, *e.g.*, [62, 167, 168, 21].

Affine addition. If $f(x) = \varphi(x) + a^Tx + b$, then

$$\mathbf{prox}_{\lambda f}(v) = \mathbf{prox}_{\lambda\varphi}(v - \lambda a).$$

Regularization. If $f(x) = \varphi(x) + (\rho/2)\|x - a\|_2^2$, then

$$\mathbf{prox}_{\lambda f}(v) = \mathbf{prox}_{\tilde{\lambda}\varphi}\left((\tilde{\lambda}/\lambda)v + (\rho\tilde{\lambda})a\right),$$

where $\tilde{\lambda} = \lambda/(1 + \lambda\rho)$.

2.3 Fixed points

The point $x^\star$ minimizes f if and only if

$$x^\star = \mathbf{prox}_f(x^\star),$$

i.e., if $x^\star$ is a fixed point of $\mathbf{prox}_f$. (We can consider $\lambda = 1$ without loss of generality, since $x^\star$ minimizes f if and only if it minimizes λf.) This fundamental property gives a link between proximal operators and fixed point theory; *e.g.*, many proximal algorithms for optimization can be interpreted as methods for finding fixed points of appropriate operators. This viewpoint is often useful in the analysis of these methods.

Proof. We can show directly that if $x^\star$ minimizes f, then $\mathbf{prox}_f(x^\star) = x^\star$. We assume for convenience that f is subdifferentiable on its domain, though the result is true in general.

If $x^\star$ minimizes f, *i.e.*, $f(x) \geq f(x^\star)$ for any x, then

$$f(x) + (1/2)\|x - x^\star\|_2^2 \geq f(x^\star) = f(x^\star) + (1/2)\|x^\star - x^\star\|_2^2$$

for any x, so $x^\star$ minimizes the function $f(x)+(1/2)\|x-x^\star\|_2^2$. It follows that $x^\star = \mathbf{prox}_f(x^\star)$.

To show the converse, we use the subdifferential characterization of the minimum of a convex function [171]. The point $\tilde{x}$ minimizes

$$f(x) + (1/2)\|x - v\|_2^2$$

(so $\tilde{x} = \mathbf{prox}_f(v)$) if and only if

$$0 \in \partial f(\tilde{x}) + (\tilde{x} - v),$$

where the sum is of a set and a point. Here, $\partial f(x) \subset \mathbf{R}^n$ is the *subdifferential* of f at x, defined by

$$\partial f(x) = \{y \mid f(z) \geq f(x) + y^T(z - x) \text{ for all } z \in \mathbf{dom}\, f\}. \qquad (2.3)$$

Taking $\tilde{x} = v = x^\star$, it follows that $0 \in \partial f(x^\star)$, so $x^\star$ minimizes f. $\square$

Fixed point algorithms. Since minimizers of f are fixed points of $\mathbf{prox}_f$, we can minimize f by finding a fixed point of its proximal operator. If $\mathbf{prox}_f$ were a contraction, *i.e.*, Lipschitz continuous with constant less than 1, repeatedly applying $\mathbf{prox}_f$ would find a (here, unique) fixed point. It turns out that while $\mathbf{prox}_f$ need not be a contraction (unless f is strongly convex), it does have a different property, *firm nonexpansiveness*, sufficient for fixed point iteration:

$$\|\mathbf{prox}_f(x) - \mathbf{prox}_f(y)\|_2^2 \leq (x - y)^T(\mathbf{prox}_f(x) - \mathbf{prox}_f(y))$$

for all $x,\ y \in \mathbf{R}^n$.

Firmly nonexpansive operators are special cases of *nonexpansive operators* (those that are Lipschitz continuous with constant 1). Iteration of a general nonexpansive operator need not converge to a fixed point: consider operators like $-I$ or rotations. However, it turns out that if N is nonexpansive, then the operator $T = (1-\alpha)I + \alpha N$, where $\alpha \in (0, 1)$, has the same fixed points as N and simple iteration of T will converge to a fixed point of T (and thus of N), *i.e.*, the sequence

$$x^{k+1} := (1-\alpha)x^k + \alpha N(x^k)$$

will converge to a fixed point of N. Put differently, *damped* iteration of a nonexpansive operator will converge to one of its fixed points.

Operators in the form $(1-\alpha)I + \alpha N$, where N is nonexpansive and $\alpha \in (0, 1)$, are called *α-averaged operators.* Firmly nonexpansive operators are averaged: indeed, they are precisely the $(1/2)$-averaged operators. In summary, both contractions and firm nonexpansions are subsets of the class of averaged operators, which in turn are a subset of all nonexpansive operators.

Averaged operators are useful because they satisfy some properties that are desirable in devising fixed point methods, and because they are a common parent of contractions and firm nonexpansions. For example, the class of averaged operators is closed under composition, unlike that of firm nonexpansions, *i.e.*, the composition of firmly nonexpansive operators need not be firmly nonexpansive but is always averaged. In addition, as mentioned above, simple iteration of an averaged operator will converge to a fixed point if one exists, a result known as the Krasnoselskii-Mann theorem. Explicitly, suppose T is averaged and has a fixed point. Define the iteration

$$x^{k+1} := T(x^k)$$

with arbitrary x^0. Then $\|T(x^k) - x^k\| \to 0$ as $k \to \infty$ and x^k converges to a fixed point of T [10, §5.2]; also see, *e.g.*, [135, 41, 15, 99, 61].

This immediately suggests the simplest proximal method,

$$x^{k+1} := \mathbf{prox}_{\lambda f}(x^k),$$

which is called *proximal minimization* or the *proximal point algorithm.* We discuss it in detail in §4.1; for example, it converges under the mildest possible assumption, which is simply that a minimizer exists.

2.4 Proximal average

Let $f_1, \ldots, f_m$ be closed proper convex functions. Then we have that

$$\frac{1}{m}\sum_{i=1}^{m} \mathbf{prox}_{f_i} = \mathbf{prox}_g,$$

where g is a function called the *proximal average* of $f_1, \ldots, f_m$. In other words, the average of the proximal operators of a set of functions is itself the proximal operator of some function, and this function is called the proximal average. This operator is fundamental and often appears in parallel proximal algorithms, which we discuss in Chapter 5. For example, such algorithms typically involve a step that evaluates the proximal operator of a number of functions independently in parallel and then averages the results.

The proximal average has a number of interesting properties. For example, the minimizers of g are the minimizers of the sum of the Moreau envelopes (see §3.1) of the f_i. See [12] for more discussion.

2.5 Moreau decomposition

The following relation always holds:

$$v = \mathbf{prox}_f(v) + \mathbf{prox}_{f^*}(v), \tag{2.4}$$

where

$$f^*(y) = \sup_x \left(y^T x - f(x)\right)$$

is the *convex conjugate* of f. This property, known as *Moreau decomposition*, is the main relationship between proximal operators and duality.

The Moreau decomposition can be viewed as a generalization of orthogonal decomposition induced by a subspace. If L is a subspace, then its *orthogonal complement* is

$$L^\perp = \{y \mid y^T x = 0 \text{ for all } x \in L\},$$

and we have that, for any v,

$$v = \Pi_L(v) + \Pi_{L^\perp}(v).$$

This follows from Moreau decomposition since $(I_L)^* = I_{L^\perp}$.

Similarly, when f is the indicator function of the closed convex cone K, we have that

$$v = \Pi_K(v) + \Pi_{K^\circ}(v),$$

where

$$K^\circ = \{y \mid y^T x \leq 0 \text{ for all } x \in K\}$$

is the *polar cone* of K, which is the negative of the *dual cone*

$$K^* = \{y \mid y^T x \geq 0 \text{ for all } x \in K\}.$$

Moreau decomposition gives a simple way to obtain the proximal operator of a function f in terms of the proximal operator of f^*. For example, if $f = \|\cdot\|$ is a general norm, then $f^* = I_{\mathcal{B}}$, where

$$\mathcal{B} = \{x \mid \|x\|_* \leq 1\}$$

is the unit ball for the dual norm $\|\cdot\|_*$, defined by

$$\|z\|_* = \sup\{z^T x \mid \|x\| \leq 1\}.$$

By Moreau decomposition, this implies that

$$v = \mathbf{prox}_f(v) + \Pi_{\mathcal{B}}(v).$$

In other words, we can easily evaluate $\mathbf{prox}_f$ if we know how to project onto $\mathcal{B}$ (and vice versa). This example is discussed in detail in §6.5.

3

Interpretations

Here we collect a variety of interpretations of proximal operators and discuss them in detail. They are useful for developing intuition about proximal operators and for giving interpretations of proximal algorithms. For example, we have seen that proximal operators can be viewed as a generalization of projections, and we will see that some proximal algorithms are generalizations of projection algorithms.

3.1 Moreau-Yosida regularization

The *infimal convolution* of closed proper convex functions f and g on $\mathbf{R}^n$, denoted $f \,\square\, g$, is defined as

$$(f \,\square\, g)(v) = \inf_x \left(f(x) + g(v - x) \right),$$

with $\mathbf{dom}(f \,\square\, g) = \mathbf{dom}\, f + \mathbf{dom}\, g$.

The main example relevant here is the following. Given $\lambda > 0$, the *Moreau envelope* or *Moreau-Yosida regularization* $M_{\lambda f}$ of the function λf is defined as $M_{\lambda f} = \lambda f \,\square (1/2)\|\cdot\|_2^2$, *i.e.*,

$$M_{\lambda f}(v) = \inf_x \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right). \tag{3.1}$$

This is also referred to as the Moreau envelope of f with parameter λ.

The Moreau envelope M_f is essentially a smoothed or regularized form of f: It has domain $\mathbf{R}^n$, even when f does not, and it is continuously differentiable, even when f is not. In addition, the sets of minimizers of f and M_f are the same. The problems of minimizing f and M_f are thus equivalent, and the latter is always a smooth optimization problem (with the caveat that M_f may be difficult to evaluate). Indeed, some algorithms for minimizing f are better interpreted as algorithms for minimizing M_f, as we will see.

To see why M_f is a smoothed form of f, consider that

$$(f \,\Box\, g)^* = f^* + g^*,$$

i.e., that infimal convolution is dual to addition [171, §16]. Because $M_f^{**} = M_f$ and $(1/2)\|\cdot\|_2^2$ is self-dual, it follows that

$$M_f = (f^* + (1/2)\|\cdot\|_2^2)^*.$$

In general, the conjugate φ^* of a closed proper convex function φ is smooth when φ is strongly convex. This suggests that the Moreau envelope M_f can be interpreted as obtaining a smooth approximation to a function by taking its conjugate, adding regularization, and then taking the conjugate again. With no regularization, this would simply give the original function; with the quadratic regularization, it gives a smooth approximation. For example, applying this technique to the absolute value function $|\cdot|$ gives the *Huber function*

$$\varphi^{\rm huber}(x) = \begin{cases} x^2 & |x| \le 1 \\ 2|x| - 1 & |x| > 1. \end{cases}$$

This perspective is very related to recent work by Nesterov [152]; for more on this connection, see [19].

The proximal operator and Moreau envelope of f share many relationships. For example, $\mathbf{prox}_f$ returns the (unique) point that actually achieves the infimum that defines M_f, *i.e.*,

$$M_f(x) = f(\mathbf{prox}_f(x)) + (1/2)\|x - \mathbf{prox}_f(x)\|_2^2.$$

In addition, the gradient of the Moreau envelope is given by

$$\nabla M_{\lambda f}(x) = (1/\lambda)(x - \mathbf{prox}_{\lambda f}(x)). \qquad (3.2)$$

We can rewrite this as

$$\mathbf{prox}_{\lambda f}(x) = x - \lambda \nabla M_{\lambda f}(x), \tag{3.3}$$

which shows that $\mathbf{prox}_{\lambda f}$ can be viewed as a gradient step, with step size λ, for minimizing $M_{\lambda f}$ (which has the same minimizers as f). Combining this with the Moreau decomposition (2.4) gives a formula relating the proximal operator, Moreau envelope, and the conjugate:

$$\mathbf{prox}_f(x) = \nabla M_{f^*}(x).$$

It is possible to consider infimal convolution and the Moreau envelope for nonconvex functions, in which case some, but not all, of the properties given above hold; see, *e.g.*, [163]. We limit the discussion here to the case when the functions are convex.

3.2 Resolvent of subdifferential operator

We can view the subdifferential operator ∂f, defined in (2.3), of a closed proper convex function f as a *point-to-set mapping* or a *relation* on $\mathbf{R}^n$, *i.e.*, ∂f takes each point $x \in \mathbf{dom}\, f$ to the set $\partial f(x)$. Any point $y \in \partial f(x)$ is called a *subgradient* of f at x. When f is differentiable, we have $\partial f(x) = \{\nabla f(x)\}$ for all x; we refer to the (point-to-point) mapping ∇f from $x \in \mathbf{dom}\, f$ to $\nabla f(x)$ as the *gradient mapping*.

The proximal operator $\mathbf{prox}_{\lambda f}$ and the subdifferential operator ∂f are related as follows:

$$\mathbf{prox}_{\lambda f} = (I + \lambda \partial f)^{-1}. \tag{3.4}$$

The (point-to-point) mapping $(I + \lambda \partial f)^{-1}$ is called the *resolvent* of the operator ∂f with parameter $\lambda > 0$, so the proximal operator is the resolvent of the subdifferential operator.

The resolvent formula (3.4) must be interpreted carefully. All the operators on the righthand side (scalar multiplication, sum, and inverse) are operations on relations, so $(I + \lambda \partial f)^{-1}$ is a relation. It turns out, however, that this relation has domain $\mathbf{R}^n$, is single-valued, and so is a function, even though ∂f is not.

Proof of (3.4). As before, we assume for convenience that f is subdifferentiable on its domain. By definition, if $z \in (I + \lambda \partial f)^{-1}(x)$, then

$$x \in (I + \lambda \partial f)(z) = z + \lambda \partial f(z).$$

This can be expressed as

$$0 \in \partial f(z) + (1/\lambda)(z - x),$$

which can in turn be rewritten as

$$0 \in \partial_z \left(f(z) + (1/2\lambda)\|z - x\|_2^2 \right),$$

where the subdifferential is with respect to z.

As in §2.3, this is the necessary and sufficient condition for z to minimize the strongly convex function within the parentheses above:

$$z = \underset{u}{\operatorname{argmin}} \left(f(u) + (1/2\lambda)\|u - x\|_2^2 \right).$$

This shows that $z \in (I + \lambda \partial f)^{-1}(x)$ if and only if $z = \mathbf{prox}_{\lambda f}(x)$ and, in particular, that $(I + \lambda \partial f)^{-1}$ is single-valued. □

3.3 Modified gradient step

There are several ways of interpreting the proximal operator as a gradient step for minimizing f or a function related to f. For instance, we have already seen in (3.3) that

$$\mathbf{prox}_{\lambda f}(x) = x - \lambda \nabla M_{\lambda f}(x),$$

i.e., $\mathbf{prox}_{\lambda f}$ is a gradient step for minimizing the Moreau envelope of f with step size λ. Here we discuss other similar interpretations.

If f is twice differentiable at x, with $\nabla^2 f(x) \succ 0$ (*i.e.*, with $\nabla^2 f(x)$ positive definite), then, as $\lambda \to 0$,

$$\mathbf{prox}_{\lambda f}(x) = (I + \lambda \nabla f)^{-1}(x) = x - \lambda \nabla f(x) + o(\lambda).$$

In other words, for small λ, $\mathbf{prox}_{\lambda f}(x)$ converges to a gradient step in f with step length λ. So the proximal operator can be interpreted (for small λ) as an approximation of a gradient step for minimizing f.

We now consider proximal operators of approximations to f and examine their relation to gradient (or other) steps for minimizing f. If f is differentiable, its first-order approximation near v is

$$\hat{f}_v^{(1)}(x) = f(v) + \nabla f(v)^T(x-v),$$

and if it is twice differentiable, its second-order approximation is

$$\hat{f}_v^{(2)}(x) = f(v) + \nabla f(v)^T(x-v) + (1/2)(x-v)^T \nabla^2 f(v)(x-v).$$

The proximal operator of the first-order approximation is

$$\mathbf{prox}_{\lambda \hat{f}_v^{(1)}}(v) = v - \lambda \nabla f(v),$$

which is a standard gradient step with step length λ. The proximal operator of the second-order approximation is

$$\mathbf{prox}_{\lambda \hat{f}_v^{(2)}}(v) = v - (\nabla^2 f(v) + (1/\lambda)I)^{-1}\nabla f(v).$$

The step on the righthand side is very familiar: it is a Tikhonov-regularized Newton update, also known as a *Levenberg-Marquardt update* [126, 136] or a *modified Hessian Newton update* [155]. Thus, gradient and Levenberg-Marquardt steps can be viewed as proximal operators of first and second-order approximations of f.

3.4 Trust region problem

A *trust region problem* has the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \|x-v\|_2 \le \rho, \end{array} \qquad (3.5)$$

with variable $x \in \mathbf{R}^n$, where $\rho > 0$ is the radius of the trust region. These problems typically arise when f is an approximation to or surrogate for some true objective φ that is only accurate near some point v; for example, f may be a second-order approximation to φ at v. The solution to the trust region problem then gives a search direction in some larger iterative procedure for minimizing φ.

The proximal problem

$$\text{minimize} \quad f(x) + (1/2\lambda)\|x-v\|_2^2 \qquad (3.6)$$

involves the same two functions of x, $f(x)$ and $\|x - v\|_2$, but the trust region constraint on distance from v appears as a (squared) penalty.

Roughly speaking, the two problems have the same solutions for appropriate choices of the parameters ρ and λ. More precisely, every solution of the proximal problem (3.6) is also a solution of the trust region problem (3.5) for some choice of ρ. Conversely, every solution of the trust region problem (3.5) is either an unconstrained minimizer of f or a solution of the proximal problem (3.6) for some choice of λ.

To see this, we examine the optimality conditions for (3.5) and (3.6). For the proximal problem (3.6), the optimality condition is simply

$$0 \in \partial f(x^{\mathrm{pr}}) + (1/\lambda)(x^{\mathrm{pr}} - v). \tag{3.7}$$

For the trust region problem (3.5), assuming there is no minimizer of f within the ball $\{x \mid \|x - v\|_2 \le \rho\}$, the optimality conditions are

$$0 \in \partial f(x^{\mathrm{tr}}) + \mu \frac{x^{\mathrm{tr}} - v}{\|x^{\mathrm{tr}} - v\|_2}, \qquad \|x^{\mathrm{tr}} - v\|_2 = \rho, \tag{3.8}$$

for some $\mu > 0$.

We immediately see that a solution x^{tr} of the trust region problem satisfies (3.7) when $\lambda = \rho/\mu$. Conversely, a solution x^{pr} of the proximal problem satisfies (3.8) with $\rho = \|x^{\mathrm{pr}} - v\|_2$ and $\mu = \rho/\lambda$.

3.5 Notes and references

Proximal operators took their current name and form in the 1960s in seminal work by Moreau [144, 145]. His initial focus was on interpreting proximal operators as generalized projections and on Moreau decomposition. Moreau also coined the term ‘infimal convolution’, while the more recent term ‘epi-addition’ is from the variational analysis literature [177]. The idea of the Moreau envelope (sometimes called *Moreau-Yosida regularization*) traces back to Moreau and to Yosida’s work in functional analysis [202]; see [124, 125] for some more recent work. The interpretation of a Moreau envelope as providing a regularized form of f originated with Attouch [3]. There has also been work on generalizing the idea of Moreau envelopes and proximal operators to non-quadratic penalties; see, *e.g.*, [11, 50, 19].

The relationship between proximal operators and resolvents was perhaps first discussed in Rockafellar's [176] fundamental paper on the proximal point algorithm. The key property of the subdifferential being used is that it is a *monotone operator*, so the resolvent interpretation is typically used in monotone operator theory. Monotone operator theory originated in functional analysis; see, *e.g.*, the classical work of Brézis [38], Browder [40, 39, 41, 42], Minty [141, 142], Kachurovskii [113, 114], and Rockafellar [173, 172], as well as Eckstein's thesis [80] and the recent monograph by Bauschke and Combettes [10]. Rockafellar's papers from the 1970s contain many of the main results on the role of monotone operators in optimization. This work continues to this day; see the bibliography in [10] for a thorough list of references.

The interpretation of the gradient method as a proximal method with the first-order approximation is well-known; see, *e.g.*, [164]. The other interpretations in §3.3 appear to be new.

4

Proximal Algorithms

We describe some important algorithms for solving convex optimization problems that rely on the use of proximal operators. These algorithms are very different from most methods in that the interface to the objective or constraint terms is via proximal operators, not their subgradients or derivatives.

There is a wide literature on applying various proximal algorithms to particular problems or problem domains, such as nuclear norm problems [185], max norm problems [121], sparse inverse covariance selection [180], MAP inference in undirected graphical models [170], loss minimization in machine learning [33, 75, 112, 4], optimal control [157], energy management [118], and signal processing [63].

4.1 Proximal minimization

The *proximal minimization algorithm*, also called *proximal iteration* or the *proximal point algorithm*, is

$$x^{k+1} := \mathbf{prox}_{\lambda f}(x^k), \tag{4.1}$$

where $f : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is a closed proper convex function, k is the iteration counter, and x^k denotes the kth iterate of the algorithm.

If f has a minimum, then x^k converges to the set of minimizers of f and $f(x^k)$ converges to its optimal value [10]. A variation on the proximal minimization algorithm uses parameter values that change in each iteration; we simply replace the constant value λ with λ^k in the iteration. Convergence is guaranteed provided $\lambda^k > 0$ and $\sum_{k=1}^{\infty} \lambda^k = \infty$. Another variation allows the minimizations required in evaluating the proximal operator to be carried out with error, provided the errors in the minimizations satisfy certain conditions (such as being summable).

This basic proximal method has not found many applications. Each iteration requires us to minimize the function f plus a quadratic, so the proximal algorithm would be useful in a situation where it is hard to minimize the function f (our goal), but easy (or at least easier) to minimize f plus a quadratic. We will see one important application, iterative refinement for solving linear equations, in §4.1.2 (although iterative refinement was not originally derived from proximal minimization). A related application, mentioned below, is in solving ill-conditioned smooth minimization problems using an iterative solver.

4.1.1 Interpretations

The proximal minimization algorithm can be interpreted many ways. One simple perspective is that it is the standard gradient method applied to the Moreau envelope M_f rather than f (see (3.3)). Another is that it is simple iteration for finding a fixed point of $\mathbf{prox}_{\lambda f}$, which works because $\mathbf{prox}_{\lambda f}$ is firmly nonexpansive (see §2.3). We now present additional interpretations that require some more discussion.

Disappearing Tikhonov regularization. Another simple interpretation is as quadratic (Tikhonov) regularization that 'goes away' in the limit. In each step we solve the regularized problem

$$\text{minimize} \quad f(x) + (1/2\lambda)\|x - x^k\|_2^2.$$

The second term can be interpreted as quadratic (Tikhonov) regularization centered at the previous iterate x^k; in other words, it is a damping term that encourages x^{k+1} not to be very far from x^k.

Suppose that f is smooth and that we use an iterative method to solve this subproblem, such as a gradient or conjugate gradient method. For such methods, this (sub)problem becomes easier as more quadratic regularization is added, *i.e.*, the smaller λ is. Here, 'easier' can mean fewer iterations, faster convergence, or higher reliability. (One method for choosing λ^k is to take it small enough to make the subproblem easy enough to solve in, say, ten iterations of some method.)

As the proximal algorithm converges, x^{k+1} gets close to x^k, so the effect of the quadratic regularization goes to zero, in the sense that the quadratic regularization contributes a term to the gradient that decreases to zero as the algorithm proceeds.

In this case, we can think of the proximal minimization method as a principled way to introduce quadratic regularization into a smooth minimization problem in order to improve convergence of some iterative method in such a way that the final result obtained is not affected by the regularization. This is done by shifting the 'center' of the regularization to the previous iterate.

Gradient flow. Proximal minimization can be interpreted as a discretized method for solving a differential equation whose equilibrium points are the minimizers of a differentiable convex function f. The differential equation

$$\frac{d}{dt}x(t) = -\nabla f(x(t)), \tag{4.2}$$

with variable $x : \mathbf{R}_+ \to \mathbf{R}^n$, is called the *gradient flow* for f. (Here $\mathbf{R}_+$ denotes the nonnegative reals $\{t \in \mathbf{R} \mid t \geq 0\}$.) The equilibrium points of the gradient flow are the zeros of ∇f, which are exactly the minimizers of f.

We can think of the gradient flow as a continuous-time analog of the gradient method for minimizing f. The gradient flow solves the problem of minimizing f in the sense that for every trajectory x of the gradient flow, we have $f(x(t)) \to p^\star$, where $p^\star$ is the minimum of f. To minimize f, then, we start from any initial vector $x(0)$ and (numerically) trace its trajectory as $t \to \infty$.

The idea of the gradient flow can be generalized to cases where f

is not differentiable via the *subgradient differential inclusion*

$$\frac{d}{dt}x(t) \in -\partial f(x(t)).$$

For simplicity, our discussion will stick to the differentiable case.

With a small abuse of notation, let x^k be the approximation of $x(kh)$, where $h > 0$ is a small step size. We compute x^k by discretizing the differential equation (4.2), *i.e.*, by numerical integration.

The simplest discretization of (4.2) is

$$\frac{x^{k+1} - x^k}{h} = -\nabla f(x^k), \tag{4.3}$$

known as the *forward Euler discretization*. Here, the derivative of x at time $t = kh$ is replaced by the *divided difference* looking forward in time over the interval $[kh, (k+1)h]$, *i.e.*,

$$\frac{x((k+1)h) - x(kh)}{(k+1)h - kh}.$$

To obtain an algorithm, we solve (4.3) for the next iterate x^{k+1}, giving the iteration

$$x^{k+1} := x^k - h\nabla f(x^k).$$

This is the standard gradient descent iteration with step size h. Thus, the gradient descent method can be interpreted as the forward Euler method for numerical integration applied to the gradient flow.

The *backward Euler method* uses the discretization

$$\frac{x^{k+1} - x^k}{h} = -\nabla f(x^{k+1}),$$

where we replace the derivative at time $t = (k+1)h$ by the divided difference looking *backward* over the interval $[kh, (k+1)h]$. This method is known to have better approximation properties than forward Euler, especially for differential equations that converge, as the gradient flow does. Its main disadvantage is that it cannot be rewritten as an iteration that gives x^{k+1} explicitly in terms of x^k. For this reason, it is called an *implicit method*, in contrast to *explicit methods* like forward Euler.

To find x^{k+1}, we solve the equation

$$x^{k+1} + h\nabla f(x^{k+1}) = x^k,$$

which, by (3.4), is equivalent to

$$x^{k+1} = \mathbf{prox}_{hf}(x^k).$$

Thus, the proximal minimization method is the backward Euler method for numerical integration applied to the gradient flow differential equation. The parameter λ in the standard proximal minimization method corresponds to the time step used in the discretization.

This interpretation suggests that the method should work, given enough assumptions on ∇f and perhaps assuming that λ is small. In fact, we know more from the other analyses; in particular, we know that the proximal method works, exactly, for any positive λ, even when the function f is not differentiable or finite.

In this section, we saw that gradient steps (in optimization) correspond to forward Euler steps (in solving the gradient flow differential equation) and backward Euler steps correspond to proximal steps. In the sequel, we often refer to gradient steps as *forward steps* and proximal steps as *backward steps*.

4.1.2 Iterative refinement

We now discuss a special case of the proximal minimization algorithm that is well-known in numerical linear algebra and is based on the idea of asymptotically disappearing Tikhonov regularization.

Consider the problem of minimizing the quadratic function

$$f(x) = (1/2)x^T Ax - b^T x,$$

where $A \in \mathbf{S}^n_+$ (the set of symmetric positive semidefinite $n \times n$ matrices). This problem is, of course, equivalent to solving the system of linear equations $Ax = b$, and when A is nonsingular, the unique solution is $x = A^{-1}b$. This problem arises in many applications, ranging from least squares fitting to the numerical solution of elliptic PDEs.

The proximal operator for f at x^k can be expressed analytically:

$$\begin{array}{rcl} \mathbf{prox}_{\lambda f}(x^k) & = & \underset{x}{\operatorname{argmin}} \left((1/2)x^T Ax - b^T x + (1/2\lambda)\|x - x^k\|_2^2\right) \\ & = & (A + (1/\lambda)I)^{-1}(b + (1/\lambda)x^k). \end{array}$$

The proximal minimization method is then

$$x^{k+1} := (A + (1/\lambda)I)^{-1}(b + (1/\lambda)x^k),$$

which can be rewritten as

$$x^{k+1} := x^k + (A + \epsilon I)^{-1}(b - Ax^k), \tag{4.4}$$

where $\epsilon = 1/\lambda$. We know that this converges to a solution of $Ax = b$ (provided one exists) as long as $\lambda > 0$ (which is the same as $\epsilon > 0$). The algorithm (4.4) is a standard algorithm, called *iterative refinement*, for solving $Ax = b$ using only the regularized inverse $(A + \epsilon I)^{-1}$ [98, 143, 139]. The second term on the righthand side of (4.4) is called the *correction* or *refinement* to the approximate solution x^k.

Iterative refinement is useful in the following situation. Suppose that A is singular or has very high condition number. In this case, we cannot solve $Ax = b$ by computing a Cholesky factorization of A, either because the factorization does not exist or because it cannot be computed stably. However, the Cholesky factorization of the regularized matrix $A + \epsilon I$ always exists (because this matrix is positive definite) and can be stably computed (assuming its condition number is not huge). Iterative refinement is an iterative method for solving $Ax = b$ using the Cholesky factorization of $A + \epsilon I$.

Iterative refinement is usually described as follows. Since A^{-1} need not exist (and if it exists, it may be huge), we prefer to approximately solve $Ax = b$ using $\hat{A}^{-1} = (A + \epsilon I)^{-1}$ instead. If ϵ is small, so $A \approx \hat{A}$, our first guess would be $x^1 = \hat{A}^{-1}b$, which has residual $r^1 = b - Ax^1$. We then compute a correction term δ^1 so that $x^2 = x^1 + \delta^1$ is a better approximation than x^1. The perfect correction would be $\delta^1 = A^{-1}r^1$, which is obtained by solving $A(x^1 + \delta^1) = b$ for δ^1. Since we cannot use A^{-1}, we instead set $\delta^1 = \hat{A}^{-1}r^1$ and let $x^2 = x^1 + \delta^1$.

These two steps are repeated for as many iterations as needed, which in practice is typically just a few. Since this method is a special case of proximal minimization, we can conclude that iterative refinement always works (asymptotically), even when ϵ is large.

4.2 Proximal gradient method

Consider the problem

$$\text{minimize} \quad f(x) + g(x), \tag{4.5}$$

where $f : \mathbf{R}^n \to \mathbf{R}$ and $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex and f is differentiable. (Since g can be extended-valued, it can be used to encode constraints on the variable x.) In this form, we *split* the objective into two terms, one of which is differentiable. This splitting is not unique, so different splittings lead to different implementations of the proximal gradient method for the same original problem.

The *proximal gradient method* is

$$x^{k+1} := \mathbf{prox}_{\lambda^k g}(x^k - \lambda^k \nabla f(x^k)), \tag{4.6}$$

where $\lambda^k > 0$ is a step size.

When ∇f is Lipschitz continuous with constant L, this method can be shown to converge with rate $O(1/k)$ when a fixed step size $\lambda^k = \lambda \in (0, 1/L]$ is used. (As discussed in [63], the method will actually converge for step sizes smaller than $2/L$, not just $1/L$, though for step sizes larger than $1/L$, the method is no longer a 'majorization-minimization method' as discussed in the next section.) If L is not known, the step sizes λ^k can be found by a line search [18, §2.4.3]; that is, their values are chosen in each step.

Many types of line search work, but one simple one due to Beck and Teboulle [18] is the following.

```
**given** $x^k$, $\lambda^{k-1}$, and parameter $\beta \in (0,1)$.
Let $\lambda := \lambda^{k-1}$.
**repeat**
    1. Let $z := \mathbf{prox}_{\lambda g}(x^k - \lambda \nabla f(x^k))$.
    2. **break if** $f(z) \le \hat{f}_\lambda(z, x^k)$.
    3. Update $\lambda := \beta\lambda$.
**return** $\lambda^k := \lambda$, $x^{k+1} := z$.
```

The function $\hat{f}_\lambda$ is easy to evaluate; its definition is given in (4.7) and discussed further below. A typical value for the line search parameter β is $1/2$.

Special cases. The proximal gradient method reduces to other well-known algorithms in various special cases. When $g = I_{\mathcal{C}}$, $\mathbf{prox}_{\lambda g}$ is projection onto $\mathcal{C}$, in which case (4.6) reduces to the *projected gradient method* [26]. When $f = 0$, then it reduces to proximal minimization, and when $g = 0$, it reduces to the standard gradient descent method.

4.2.1 Interpretations

The first two interpretations given below are due to Beck and Teboulle [18]; we have repeated their discussion here for completeness. In the context of image processing problems, the majorization-minimization interpretation appeared in some even earlier papers by Figueiredo et al. [87, 85]. We also mention that in some special cases, additional interpretations are possible; for example, applying the method to the lasso can be interpreted as a kind of EM algorithm [86].

Majorization-minimization. We first interpret the proximal gradient method as an example of a *majorization-minimization algorithm*, a large class of algorithms that includes the gradient method, Newton's method, and the EM algorithm as special cases; see, *e.g.*, [108].

A majorization-minimization algorithm for minimizing a function $\varphi : \mathbf{R}^n \to \mathbf{R}$ consists of the iteration

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\, \hat{\varphi}(x, x^k),$$

where $\hat{\varphi}(\cdot, x^k)$ is a convex upper bound to φ that is tight at x^k, *i.e.*, $\hat{\varphi}(x, x^k) \geq \varphi(x)$ and $\hat{\varphi}(x, x) = \varphi(x)$ for all x. The reason for the name should be clear: such algorithms involve iteratively majorizing (upper bounding) the objective and then minimizing the majorization.

For an upper bound of f, consider the function $\hat{f}_\lambda$ given by

$$\hat{f}_\lambda(x, y) = f(y) + \nabla f(y)^T (x - y) + (1/2\lambda)\|x - y\|_2^2, \tag{4.7}$$

with $\lambda > 0$. For fixed y, this function is convex, satisfies $\hat{f}_\lambda(x, x) = f(x)$, and is an upper bound on f when $\lambda \in (0, 1/L]$, where L is a Lipschitz constant of ∇f. The algorithm

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\, \hat{f}_\lambda(x, x^k)$$

is thus a majorization-minimization algorithm; in fact, a little algebra shows that this algorithm is precisely the standard gradient method for minimizing f. Intuitively, we replace f with its first-order approximation regularized by a trust region penalty (see §3.4).

It then follows that the function q_λ given by

$$q_\lambda(x, y) = \hat{f}_\lambda(x, y) + g(x) \tag{4.8}$$

is similarly a surrogate for $f + g$ (with fixed y) when $\lambda \in (0, 1/L]$. The majorization-minimization algorithm

$$x^{k+1} := \operatorname*{argmin}_x q_\lambda(x, x^k)$$

can be shown to be equivalent to the proximal gradient iteration (4.6).

Another way to express the problem of minimizing $q_\lambda(x, x^k)$ is as

$$\text{minimize} \quad (1/2)\|x - (x^k - \lambda\nabla f(x^k))\|_2^2 + \lambda g(x).$$

This formulation shows that the solution x^{k+1} can be interpreted as trading off minimizing g and being close to the standard gradient step $x^k - \lambda\nabla f(x^k)$, with the trade-off determined by the parameter λ.

Fixed point iteration. The proximal gradient algorithm can also be interpreted as a fixed point iteration. A point $x^\star$ is a solution of (4.5), *i.e.*, minimizes $f + g$, if and only if

$$0 \in \nabla f(x^\star) + \partial g(x^\star).$$

For any $\lambda > 0$, this optimality condition holds if and only if the following equivalent statements hold:

$$\begin{array}{rcl} 0 & \in & \lambda\nabla f(x^\star) + \lambda\partial g(x^\star) \\ 0 & \in & \lambda\nabla f(x^\star) - x^\star + x^\star + \lambda\partial g(x^\star) \\ (I + \lambda\partial g)(x^\star) & \ni & (I - \lambda\nabla f)(x^\star) \\ x^\star & = & (I + \lambda\partial g)^{-1}(I - \lambda\nabla f)(x^\star) \\ x^\star & = & \mathbf{prox}_{\lambda g}(x^\star - \lambda\nabla f(x^\star)). \end{array}$$

The last two expressions hold with equality and not just containment because the proximal operator is single-valued, as mentioned in §3.2.

The final statement says that $x^\star$ minimizes $f + g$ if and only if it is a fixed point of the *forward-backward operator*

$$(I + \lambda\partial g)^{-1}(I - \lambda\nabla f).$$

The proximal gradient method repeatedly applies this operator to obtain a fixed point and thus a solution to the original problem. The condition $\lambda \in (0, 1/L]$, where L is a Lipschitz constant of ∇f, guarantees that the forward-backward operator is averaged and thus that the iteration converges to a fixed point (when one exists).

Forward-backward integration of gradient flow. The proximal gradient algorithm can be interpreted using gradient flows. Here, the gradient flow system (4.2) takes the form

$$\frac{d}{dt}x(t) = -\nabla f(x(t)) - \nabla g(x(t)),$$

assuming here that g is also differentiable.

To obtain a discretization of (4.2), we replace the derivative on the lefthand side with the difference $(x^{k+1} - x^k)/h$. We also replace the value $x(t)$ on the righthand side with either x^k (giving the forward Euler discretization) or x^{k+1} (giving the backward Euler discretization). It is reasonable to use either x^k or x^{k+1} on the righthand side since h is supposed to be a small step size, so $x(kh)$ and $x((k+1)h)$ should not be too different. Indeed, it is possible to use *both* x^k and x^{k+1} on the righthand side to replace different occurrences of $x(t)$. The resulting discretizations lead to algorithms known as *operator splitting methods*.

For example, we can consider the discretization

$$\frac{x^{k+1} - x^k}{h} = -\nabla f(x^k) - \nabla g(x^{k+1}),$$

where we replace $x(t)$ in the argument to f with the forward value x^k, and we replace $x(t)$ in the argument to g with the backward value x^{k+1}. Rearranging, this gives the update

$$x^{k+1} := (I + h\nabla g)^{-1}(I - h\nabla f)x^k,$$

This is known as *forward-backward splitting* and is exactly the proximal gradient iteration (4.6) when $\lambda = h$. In other words, the proximal gradient method can be interpreted as a method for numerically integrating the gradient flow differential equation that uses a forward Euler step for the differentiable part f and a backward Euler step for the (possibly) nondifferentiable part g.

4.3 Accelerated proximal gradient method

So-called 'accelerated' versions of the basic proximal gradient algorithm include an extrapolation step in the algorithm. One simple version is

$$\begin{aligned} y^{k+1} &:= x^k + \omega^k (x^k - x^{k-1}) \\ x^{k+1} &:= \mathbf{prox}_{\lambda^k g}(y^{k+1} - \lambda^k \nabla f(y^{k+1})) \end{aligned}$$

where $\omega^k \in [0, 1)$ is an extrapolation parameter and λ^k is the usual step size. (We let $\omega^0 = 0$, so the value x^{-1} appearing in the first extra step doesn't matter.) These parameters must be chosen in specific ways to achieve the convergence acceleration. One simple choice [194] takes

$$\omega^k = \frac{k}{k+3}.$$

It remains to choose the step sizes λ^k. When ∇f is Lipschitz continuous with constant L, this method can be shown to converge in objective value with rate $O(1/k^2)$ when a fixed step size $\lambda^k = \lambda \in (0, 1/L]$ is used. If L is not known, the step sizes λ^k can be found by a line search [18]; that is, their values are chosen in each step.

Many types of line search work, but one simple one due to Beck and Teboulle [18] is the following.

```
given $y^k$, $\lambda^{k-1}$, and parameter $\beta \in (0, 1)$.
Let $\lambda := \lambda^{k-1}$.
repeat
    1. Let $z := \mathbf{prox}_{\lambda g}(y^k - \lambda \nabla f(y^k))$.
    2. break if $f(z) \le \hat{f}_\lambda(z, y^k)$.
    3. Update $\lambda := \beta\lambda$.
return $\lambda^k := \lambda$, $x^{k+1} := z$.
```

As before, the function $\hat{f}_\lambda$ is defined in (4.7). The line search here is the same as in the standard proximal gradient method, except that it uses the extrapolated value y^k rather than x^k.

Following Nesterov, this is called an *accelerated* or *optimal* first-order method because it has a worst-case convergence rate that is superior to the standard method and that cannot be improved further [149, 150]. There are several versions of such methods, such as in Nesterov [153] and Tseng [190]; the software package TFOCS [22] is based on and contains several implementations of such methods.

4.4 Alternating direction method of multipliers

Consider the problem

$$\text{minimize} \quad f(x) + g(x)$$

where $f,\ g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex functions. (In this splitting, both f and g can be nonsmooth.) Then the *alternating direction method of multipliers* (ADMM), also known as *Douglas-Rachford splitting*, is

$$\begin{array}{rcl} x^{k+1} & := & \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} & := & \mathbf{prox}_{\lambda g}(x^{k+1} + u^k) \\ u^{k+1} & := & u^k + x^{k+1} - z^{k+1}, \end{array}$$

where k is an iteration counter. This method converges under more or less the most general possible conditions; see [33, §3.2] for details.

While x^k and z^k converge to each other, and to optimality, they have slightly different properties. For example, $x^k \in \mathbf{dom}\, f$ while $z^k \in \mathbf{dom}\, g$, so if g encodes constraints, the iterates z^k satisfy the constraints, while the iterates x^k satisfy the constraints only in the limit. If $g = \|\cdot\|_1$, then z^k will be sparse because $\mathbf{prox}_{\lambda g}$ is soft thresholding (see (6.9)), while x^k will only be close to z^k (close to sparse).

The advantage of ADMM is that the objective terms (which can both include constraints, since they can take on infinite values) are handled completely separately, and indeed, the functions are accessed only through their proximal operators. ADMM is most useful when the proximal operators of f and g can be efficiently evaluated but the proximal operator for $f + g$ is not easy to evaluate.

Special cases. When g is the indicator function of a closed convex set $\mathcal{C}$, its proximal operator $\mathbf{prox}_{\lambda g}$ reduces to projection onto $\mathcal{C}$. In this case, ADMM is a method for solving the generic convex constrained problem of minimizing f over $\mathcal{C}$ that only uses the proximal operator of the objective and projection onto the constraint set. (We can reverse the roles, with f the indicator function of $\mathcal{C}$, and g a generic convex function; this gives a slightly different algorithm.)

As a further specialization, suppose that f is the indicator function of a closed convex set $\mathcal{C}$ and g is the indicator function of a closed convex set $\mathcal{D}$. The problem of minimizing $f + g$ is then equivalent to the convex feasibility problem of finding a point $x \in \mathcal{C} \cap \mathcal{D}$. Both proximal operators reduce to projections, so the ADMM algorithm for this problem becomes

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

The parameter λ does not appear in this algorithm because both proximal operators are projections. This algorithm is similar to, but not the same as, *Dykstra's alternating projections method* [79, 8]. (In [33], we erroneously claimed that the two were equivalent; we thank Heinz Bauschke for bringing this error to our attention and clarifying the point in [13].)

Like the classical method of alternating projections due to von Neumann [198], this method requires one projection onto each set in each iteration. However, its convergence is usually much faster in practice.

4.4.1 Interpretations

Integral control of a dynamical system. The first two steps in ADMM can be viewed as a discrete-time dynamical system with state z and input or control u, *i.e.*, z^{k+1} is a function of x^k and u^k. The goal is to choose u to achieve $x = z$, so the residual $x^{k+1} - z^{k+1}$ can be viewed as an error signal. The u-update in ADMM shows that u^k is the running sum of the errors, which is the discrete-time analogue of the running integral of an error signal. Thus ADMM can be viewed as a classical *integral control method* [88] for driving an error signal to zero by feeding back the integral of the error to its input.

Augmented Lagrangians. One important interpretation relies on the idea of an *augmented Lagrangian.* We first write the problem of minimizing $f(x) + g(x)$ as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array} \tag{4.9}$$

which is called *consensus form.* Here, the variable has been split into two variables x and z, and we have added the *consensus constraint* that they must agree. This is evidently equivalent to minimizing $f + g$.

The *augmented Lagrangian* associated with the problem (4.9) is

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

where $\rho > 0$ is a parameter and $y \in \mathbf{R}^n$ is a dual variable associated with the consensus constraint. This is the usual Lagrangian augmented with an additional quadratic penalty on the equality constraint function. ADMM can then be expressed as

$$\begin{array}{rcl} x^{k+1} & := & \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \\ z^{k+1} & := & \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} & := & y^k + \rho(x^{k+1} - z^{k+1}). \end{array}$$

In each of the x and z steps, L_ρ is minimized over the variable, using the most recent value of the other primal variable and the dual variable. The dual variable is the (scaled) running sum of the consensus errors.

To see how the augmented Lagrangian form of ADMM reduces to the proximal version, we start from

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

and then pull the linear terms into the quadratic ones to get

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

With $u^k = (1/\rho)y^k$ and $\lambda = 1/\rho$, this is the proximal form of ADMM.

Flow interpretation. ADMM can also be interpreted as a method for solving a particular system of ordinary differential equations. Assuming for simplicity that f and g are differentiable, the optimality conditions for (4.9) are

$$\nabla f(x) + \nu = 0, \quad \nabla g(z) - \nu = 0, \quad x - z = 0, \tag{4.10}$$

where $\nu \in \mathbf{R}^n$ is a dual variable. Now consider the differential equation

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} -\nabla f(x(t)) - \rho u(t) - \rho r(t) \\ -\nabla g(z(t)) + \rho u(t) + \rho r(t) \end{bmatrix}, \quad \frac{d}{dt}u(t) = \rho r(t), \tag{4.11}$$

where $r(t) = x(t) - z(t)$ is the primal (consensus) residual and $\rho > 0$. The functions in the differential equation are the primal variables x and z, and the dual variable u. This differential equation does not have a standard name, but we will call it the *saddle point flow* for the problem (4.9), since it can be interpreted as a continuous analog of some saddle point algorithms.

It is easy to see that the equilibrium points of the saddle point flow (4.11) are the same as the optimality conditions (4.10) when $\nu = \rho u$. It can also be shown that all trajectories of the saddle point flow converge to an equilibrium point (assuming there exist $x^\star$ and $\nu^\star$ satisfying the optimality conditions). It follows that we can solve the problem (4.9) by following any trajectory of the flow (4.11) using numerical integration.

With x^k, z^k, and u^k denoting our approximations of $x(t)$, $z(t)$, and $u(t)$ at $t = kh$, where $h > 0$ is the step length, we use the discretization of (4.11) given by

$$\begin{aligned} \frac{x^{k+1} - x^k}{h} &= -\nabla f(x^{k+1}) - \rho(x^k - z^k + u^k) \\ \frac{z^{k+1} - z^k}{h} &= -\nabla g(z^{k+1}) + \rho(x^{k+1} - z^k + u^k) \\ \frac{u^{k+1} - u^k}{h} &= \rho(x^{k+1} - z^{k+1}). \end{aligned}$$

As in forward-backward splitting, we make very specific choices on the righthand side as to whether each time argument t is replaced with kh (forward) or $(k+1)h$ (backward) values. Choosing $h = \lambda$ and $\rho = 1/\lambda$, this discretization reduces directly to the proximal form of ADMM.

Fixed point iteration. ADMM can be viewed as a fixed point iteration for finding a point $x^\star$ satisfying the optimality condition

$$0 \in \partial f(x^\star) + \partial g(x^\star). \tag{4.12}$$

Fixed points x, z, u of the ADMM iteration satisfy

$$x = \mathbf{prox}_{\lambda f}(z - u), \quad z = \mathbf{prox}_{\lambda g}(x + u), \quad u = u + x - z.$$

From the last equation we conclude $x = z$, so

$$x = \mathbf{prox}_{\lambda f}(x - u), \quad x = \mathbf{prox}_{\lambda g}(x + u),$$

which can be written as

$$x = (I + \lambda \partial f)^{-1}(x - u), \quad x = (I + \lambda \partial g)^{-1}(x + u).$$

This is the same as

$$x - u \in x + \lambda \partial f(x), \quad x + u \in x + \lambda \partial g(x).$$

Adding these two equations shows that x satisfies the optimality condition (4.12). Thus, any fixed point of the ADMM iteration satisfies $x = z$, with x optimal. Convergence of the ADMM iteration to a fixed point can be established several ways; one way is to show that it is equivalent to iteration of a firmly nonexpansive operator [80].

4.4.2 Linearized ADMM

A variation of ADMM can be useful for solving problems of the form

$$\text{minimize} \quad f(x) + g(Ax),$$

where $f : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ and $g : \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$ are closed proper convex and $A \in \mathbf{R}^{m \times n}$. The only difference from the form used in standard ADMM is the presence of the matrix A in the second term.

This problem can be solved with standard ADMM by defining $\tilde{g}(x) = g(Ax)$ and minimizing $f(x) + \tilde{g}(x)$. However, this approach requires evaluation of the proximal operator of $\tilde{g}$, which is complicated by the presence of A, even when the proximal operator of g is easy to evaluate. (There are a few special cases where $\mathbf{prox}_{\tilde{g}}$ is in fact simple to evaluate; see §2.2.) The *linearized ADMM algorithm* solves the problem above using only the proximal operators of f and g and multiplication by A and A^T; in particular, g and A are handled separately.

Linearized ADMM has the form

$$\begin{array}{rcl} x^{k+1} & := & \mathbf{prox}_{\mu f}(x^k - (\mu/\lambda)A^T(Ax^k - z^k + u^k)) \\ z^{k+1} & := & \mathbf{prox}_{\lambda g}(Ax^{k+1} + u^k) \\ u^{k+1} & := & u^k + Ax^{k+1} - z^{k+1}, \end{array}$$

where the algorithm parameters λ and μ satisfy $0 < \mu \leq \lambda/\|A\|_2^2$. This reduces to standard ADMM when $A = I$ and $\mu = \lambda$.

The reason for the name is the following. Consider the problem

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

with variables x and z. The augmented Lagrangian for this problem is

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

where $y \in \mathbf{R}^m$ is a dual variable and $\rho = 1/\lambda$. In linearized ADMM, we modify the usual x-update by replacing $(\rho/2)\|Ax - z^k\|_2^2$ with

$$\rho(A^T Ax^k - A^T z^k)^T x + (\mu/2)\|x - x^k\|_2^2,$$

i.e., we linearize the quadratic term and add new quadratic regularization. The result can be expressed as a proximal operator as above.

This algorithm is discussed in many papers; see, *e.g.*, [207] or [159] and references therein. In the image processing literature, it is known as the *split inexact Uzawa method* [82, 207, 206, 106].

4.5 Notes and references

The initial work on the proximal minimization algorithm is due to Martinet [137, 138]. Proximal minimization was extended to the general *proximal point algorithm* for finding the zero of an arbitrary maximal monotone operator by Rockafellar [176]; its convergence theory has been extended in much subsequent work, *e.g.*, [132, 102, 84]. Proximal minimization is closely related to multiplier methods [117, 24, 25] and the literature on augmented Lagrangians [174, 175, 80].

The general form of forward-backward splitting was perhaps first discussed by Bruck [43]. Forward-backward splitting is an example of an *operator splitting method*, a term coined by Lions and Mercier [131]. Important papers on forward-backward splitting include those by Passty [161], Lions and Mercier [131], Fukushima and Mine [90], Gabay [92], Lemaire [122], Eckstein [80], Chen [56], Chen and Rockafellar [57], Tseng [186, 187, 189], Combettes and Wajs [64], and Beck and Teboulle [17, 18]. Relationships between proximal gradient, coordinate descent, and gradient methods are discussed in [26]. For particular problems, such as the lasso, it is possible to prove additional stronger results about the performance of the proximal gradient method [104, 69, 60, 36].

Accelerated proximal gradient methods trace their roots back to the literature on *optimal first-order methods*. The first of these was due to Nesterov [150], and there has been a substantial literature on other optimal-order algorithms since then, such as the papers by Nesterov [150, 151, 152, 153], Tseng [190], Beck and Teboulle [17, 18], Becker et al. [20, 22], Goldfarb and Scheinberg [97, 179], Güler [103], O'Donoghue and Candès [156], and many others. We note that the convergence theory of accelerated proximal gradient methods is not based on operator splitting, unlike the basic method. Finally, there are ways to accelerate the basic proximal gradient method other than the method we showed, such as through the use of Barzilai-Borwein step sizes [6, 201] or with other types of extrapolation steps [28].

ADMM is equivalent to an operator splitting method called *Douglas-Rachford splitting*, which was introduced in the 1950s for the numerical solution of partial differential equations [77]. It was first introduced in its modern form by Gabay and Mercier [93] and Glowinski and Marrocco [96] in the 1970s. See Boyd et al. [33] for a recent survey of the algorithm and its applications, including a detailed bibliography and many other references. See [199] for a recent paper on applying ADMM to solving semidefinite programming problems.

The idea of viewing optimization algorithms, or at least gradient methods, from the perspective of numerical methods for ordinary differential equations appears to originate in the 1950s [2]. These ideas were also explored by Polyak [165] and Bruck [43] in the 1970s. The interpretation of a proximal operator as a backward Euler step is well known; see, *e.g.*, Lemaire [123] and Eckstein [80] and references therein.

We also note that there are a number of less widely used proximal algorithms building on the basic methods discussed in this chapter; see, for example, [109, 91, 166, 119, 9, 188, 189, 31]. There are also closely related methods focused on specific application domains, such as [52].

Finally, the basic ideas have been generalized in various ways:

1. *Non-quadratic penalties.* Some authors have studied generalized proximal operators that use non-quadratic penalty terms, such as entropic penalties [183] and Bregman divergences [37, 50, 81, 154]. These can be used in generalized forms of proximal algorithms like the ones discussed in this chapter. For example, the mirror descent algorithm can be viewed as such a method [149, 16].

2. *Nonconvex optimization.* Some have studied proximal operators and algorithms in the nonconvex case [90, 115, 162].

3. *Infinite dimensions.* Building on Rockafellar's work, there is a substantial literature studying the proximal point algorithm in the monotone operator setting; this is closely connected to the literature on set-valued mappings, fixed point theory, nonexpansive mappings, and variational inequalities [204, 38, 105, 177, 83, 45, 10]; the recent paper by Combettes [61] is worth highlighting.

5

Parallel and Distributed Algorithms

In this chapter we describe a simple method to obtain parallel and distributed proximal algorithms for solving convex optimization problems. The method is based on the ADMM algorithm described in §4.4, and the key is to split the objective (and constraints) into two terms, at least one of which is separable. The separability of the terms gives us the ability to evaluate the proximal operator in parallel. It is also possible to construct parallel and distributed algorithms using the proximal gradient or accelerated proximal gradient methods, but this approach imposes differentiability conditions on part of the objective.

5.1 Problem structure

Let $[n] = \{1, ..., n\}$. Given $c \subseteq [n]$, let $x_c \in \mathbf{R}^{|c|}$ denote the subvector of $x \in \mathbf{R}^n$ referenced by the indices in c. The collection $\mathcal{P} = \{c_1, \dots, c_N\}$, where $c_i \subseteq [n]$, is a *partition* of $[n]$ if $\bigcup \mathcal{P} = [n]$ and $c_i \cap c_j = \emptyset$ for $i \neq j$. A function $f : \mathbf{R}^n \to \mathbf{R}$ is said to be $\mathcal{P}$*-separable* if

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

where $f_i : \mathbf{R}^{|c_i|} \to \mathbf{R}$ and x_{c_i} is the subvector of x with indices in c_i. We refer to c_i as the *scope* of f_i. In other words, f is a sum of terms f_i, each of which depends only on part of x; if each $c_i = \{i\}$, then f is fully separable. Separability is of interest because if f is $\mathcal{P}$-separable, then $(\mathbf{prox}_f(v))_i = \mathbf{prox}_{f_i}(v_i)$, where $v_i \in \mathbf{R}^{|c_i|}$, *i.e.*, the proximal operator breaks into N smaller operations that can be carried out independently in parallel. This is immediate from the separable sum property of §2.1.

Consider the problem

$$\text{minimize} \quad f(x) + g(x), \tag{5.1}$$

where $x \in \mathbf{R}^n$ and where $f,\ g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex. (In many cases of interest, g will be the indicator function of a convex set.) We assume that f and g are $\mathcal{P}$-separable and $\mathcal{Q}$-separable, respectively, where $\mathcal{P} = \{c_1, \ldots, c_N\}$ and $\mathcal{Q} = \{d_1, \ldots, d_M\}$ are partitions of $[n]$. Writing the problem explicitly in terms of the subvectors in the partitions, the problem is

$$\text{minimize} \quad \textstyle\sum_{i=1}^N f_i(x_{c_i}) + \sum_{j=1}^M g_j(x_{d_j}), \tag{5.2}$$

where $f_i : \mathbf{R}^{|c_i|} \to \mathbf{R} \cup \{+\infty\}$ and $g_j : \mathbf{R}^{|d_j|} \to \mathbf{R} \cup \{+\infty\}$. By convention, we use i to index the f blocks and j to index the g blocks.

ADMM for the problem form (5.2) is the algorithm

$$\begin{aligned}
x_{c_i}^{k+1} &:= \mathbf{prox}_{\lambda f_i}(z_{c_i}^k - u_{c_i}^k) \\
z_{d_j}^{k+1} &:= \mathbf{prox}_{\lambda g_j}(x_{d_j}^{k+1} + u_{d_j}^k) \\
u^{k+1} &:= u^k + x^{k+1} - z^{k+1}.
\end{aligned}$$

The first step involves N updates carried out independently in parallel, each of which involves evaluating the proximal operator of one of the components f_i of f, and the second step involves M updates carried out independently in parallel, each involving the proximal operator of a component g_j of g. The final step, of course, is always trivially parallelizable. This can be visualized as in Figure 5.1, which shows two partitions of a set of variables. Here, the x-update splits into 3 parts and the z-update splits into 2 parts.

If, for instance, $\mathcal{P} = \mathcal{Q}$, then the original problem has a separable objective and is thus trivially parallelizable. On the other hand, overlaps in the two partitions, as in Figure 5.1, will lead to communication

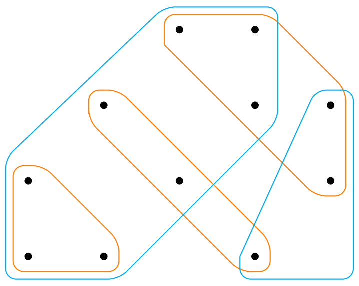

Figure 5.1: Variables are dots; the partitions $\mathcal{P}$ and $\mathcal{Q}$ are in orange and cyan.

between different subsystems. For example, if g is not separable, then the z-update will involve aggregating information across the N components that can be handled independently in the x-update. This will become more clear as we examine special cases.

5.2 Consensus

5.2.1 Global consensus

Consider the problem of minimizing an additive function, *i.e.*, a sum of terms that all share a common variable:

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

with variable $x \in \mathbf{R}^n$. The problem is to minimize each of the 'local' objectives f_i, each of which depends on the same global variable x. We aim to solve this problem in a way that allows each f_i to be handled in parallel by a separate processing element or subsystem.

We first transform the problem into *consensus form*:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_1 = x_2 = \cdots = x_N, \end{array} \tag{5.3}$$

with variables $x_i \in \mathbf{R}^n$, $i = 1, \dots, N$. In other words, we create N copies of the original global variable x so that the objective is now separable, but we add a *consensus* or *consistency constraint* that requires all these 'local' variables x_i to agree. This can be visualized as in Figure 5.2,

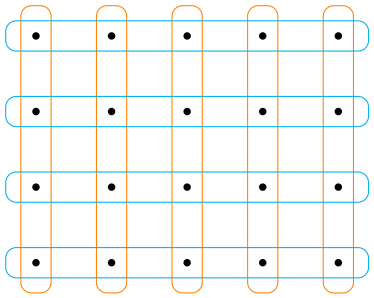

Figure 5.2: Variables are dots; the partitions $\mathcal{P}$ and $\mathcal{Q}$ are in orange and cyan.

which shows an example with $n = 4$ and $N = 5$; here, each local variable x_i is a column and the consistency constraints are drawn across rows.

The next step is to transform (5.3) into the canonical form (5.1):

$$\text{minimize} \quad \textstyle\sum_{i=1}^N f_i(x_i) + I_{\mathcal{C}}(x_1, \ldots, x_N), \tag{5.4}$$

where $\mathcal{C}$ is the *consensus set*

$$\mathcal{C} = \{(x_1, \ldots, x_N) \mid x_1 = \cdots = x_N\}. \tag{5.5}$$

In this formulation we have moved the consensus constraint into the objective using an indicator function. In the notation of (5.1), f is the sum of the terms f_i, while g is the indicator function of the consistency constraint. The partitions are given by

$$\begin{aligned} \mathcal{P} &= \{[n], n + [n], 2n + [n], \ldots, (N-1)n + [n]\}, \\ \mathcal{Q} &= \{\{i, n + i, 2n + i, \ldots, (N-1)n + i\} \mid i = 1, \ldots, N\}. \end{aligned}$$

The first partition is clear since f is additive. The consensus constraint splits across its components; it can be written as a separate consensus constraint for each component. Since the full optimization variable for (5.4) is in $\mathbf{R}^{nN}$, it is easiest to view it as in Figure 5.2, in which case it is easy to see that f is separable across columns while g is separable across rows.

We now apply ADMM as above. Evaluating $\mathbf{prox}_{\lambda g}$ reduces to projecting onto the consensus set (5.5). This is simple: we replace each z_i with its average $\overline{z} = (1/N)\sum_{i=1}^N z_i$. From this we conclude that

$\sum_{i=1}^N u_i^k = 0$, which allows for some simplifications of the general algorithm above, giving the following final method:

$$\begin{array}{rcl} x_i^{k+1} & := & \mathbf{prox}_{\lambda f_i}(\overline{x}^k - u_i^k) \\ u_i^{k+1} & := & u_i^k + x_i^{k+1} - \overline{x}^{k+1}. \end{array} \tag{5.6}$$

In this *proximal consensus algorithm*, each of the N subsystems independently carries out a dual update and evaluates its local proximal operator; in between these, all the local variables x_i^k are averaged and the result is given to each subsystem. (In distributed computing frameworks like MPI, this can be implemented with an *all-reduce operator.*)

The method is very intuitive: The (scaled) dual variables u_i, which measure the deviation of x_i from the average $\overline{x}$, are independently updated to drive the variables into consensus, and quadratic regularization helps pull the variables toward their average value while still attempting to minimize each local f_i.

5.2.2 General consensus

Consider the problem

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

where $x \in \mathbf{R}^n$ and $c_i \subseteq [n]$. Here, the c_i may overlap with each other, so $\{c_1, \dots, c_N\}$ is a *cover* rather than a partition of $[n]$. In other words, the objective f consists of a sum of terms, each of which depends on some subset of components in the full global variable x. If each $c_i = [n]$, then we recover the global consensus formulation.

We introduce a copy $z_i \in \mathbf{R}^{|c_i|}$ for each x_{c_i} and transform this into the following problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(z_i) \\ \text{subject to} & (z_1, \dots, z_N) \in \mathcal{C}, \end{array} \tag{5.7}$$

where

$$\mathcal{C} = \{(z_1, \dots, z_N) \mid (z_i)_k = (z_j)_k \text{ if } k \in c_i \cap c_j\}.$$

Roughly speaking, the z_i must agree on the components that are shared.

We can visualize this as in Figure 5.3, which is interpreted exactly like Figure 5.2 but with some dots (variables) missing. In the diagram,

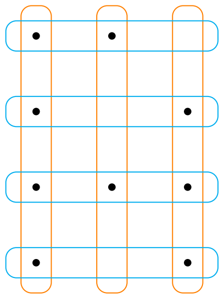

Figure 5.3: Variables are dots; the partitions $\mathcal{P}$ and $\mathcal{Q}$ are in orange and cyan.

for instance, $c_1 = \{2, 3, 4\}$, so f_1 only depends on the last three components of $x \in \mathbf{R}^n$, and $z_1 \in \mathbf{R}^3$. The consistency constraints represented by $\mathcal{C}$ say that all the variables in the same row must agree.

This problem can also be visualized using a factor graph in which the f_j are factor nodes, each individual variable component $x_i \in \mathbf{R}$ is a variable node, and an edge between x_i and f_j means that i is in scope for f_j. The example from Figure 5.3 is shown in factor graph form in Figure 5.4. There is a consensus constraint among all the variables attached to the same factor.

In the canonical form (5.1), this becomes

$$\text{minimize} \quad \textstyle\sum_{i=1}^N f_i(z_i) + I_{\mathcal{C}}(z_1, \dots, z_N). \tag{5.8}$$

As before, f is the sum of the terms f_i, while g is the indicator function of the consensus constraint. and is separable across columns. We omit the somewhat complicated explicit forms of $\mathcal{P}$ and $\mathcal{Q}$, but as before, f is separable across rows and g is separable across columns in Figure 5.3.

Applying ADMM and simplifying, we obtain the algorithm

$$\begin{array}{rcl} x_i^{k+1} & := & \mathbf{prox}_{f_i}(\overline{x}_i^k + u_i^k) \\ u_i^{k+1} & := & u_i^k + x_i^{k+1} - \overline{x}_i^{k+1}. \end{array} \tag{5.9}$$

Here,

$$(\overline{x}_i^k)_j = \frac{1}{|F_i|} \sum_{i' \in F_i} (x_{i'}^k)_j,$$

where $F_i = \{j \in [N] \mid i \in c_j\}$. Though this is complicated to define formally, it is intuitively just a 'local' averaging operator: $\overline{x}_i^k \in \mathbf{R}^{|c_i|}$ is

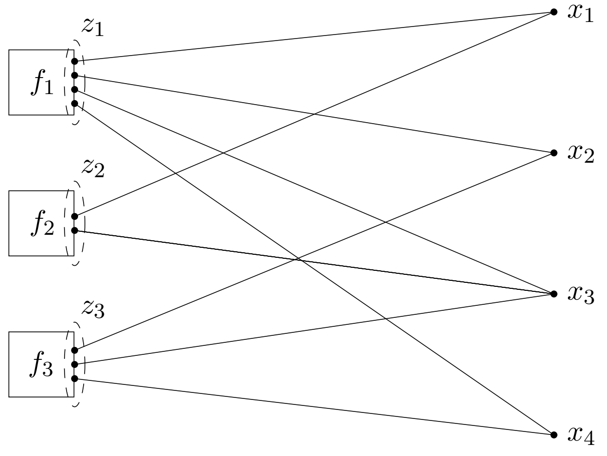

Figure 5.4: Graph form consensus optimization. Local objective terms are on the left; global variable components are on the right. Each edge in the bipartite graph is a consistency constraint, linking a local variable and a global variable component.

obtained by averaging each component only across the terms in which it is in scope. Following Figure 5.3, the variables in the same row are averaged together. This modified averaging operator shows up because the consensus set we project onto is different.

The structure of the algorithm is as before: We carry out local computations in parallel to obtain u_i^{k+1} and x_i^{k+1}, and averaging takes place in between. Since only local averaging needs to take place, this algorithm can be implemented in a completely decentralized fashion.

5.3 Exchange

5.3.1 Global exchange

The *exchange problem* is the following:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & \sum_{i=1}^N x_i = 0, \end{array} \tag{5.10}$$

with variables $x_i \in \mathbf{R}^n$, $i = 1, \ldots, N$.

The name 'exchange' comes from the following economics interpretation. The components of the vectors x_i represent quantities of commodities that are exchanged among N agents. When $(x_i)_j$ is positive, it can be viewed as the amount of commodity j *received* by agent i from the exchange. When $(x_i)_j$ is negative, its magnitude $|(x_i)_j|$ can be viewed as the amount of commodity j *contributed* by agent i to the exchange. The *equilibrium constraint* that each commodity clears is $\sum_{i=1}^N x_i = 0$, which means that the total amount of each commodity contributed by agents balances the total amount taken by agents. The exchange problem seeks the commodity quantities that minimize the *social cost*, *i.e.*, the total cost across the agents, subject to the market clearing. An optimal dual variable associated with the clearing constraint has a simple and natural interpretation as a set of equilibrium prices for the commodities.

This can be rewritten in the canonical form (5.1) as

$$\text{minimize} \quad \textstyle\sum_{i=1}^N f_i(x_i) + I_{\mathcal{C}}(x_1, \dots, x_N),$$

where $\mathcal{C}$ is the equilibrium or *clearing set*

$$\mathcal{C} = \{(x_1, \dots, x_N) \in \mathbf{R}^{nN} \mid x_1 + \cdots + x_N = 0\}. \tag{5.11}$$

This problem can be visualized exactly as before, as shown in Figure 5.2. Here, f and g are separable across rows and columns, respectively, and the definitions of $\mathcal{P}$ and $\mathcal{Q}$ are the same as before.

It remains to see how to project onto $\mathcal{C}$. This turns out to be simple de-meaning:

$$(\Pi_{\mathcal{C}}(v_1, \dots, v_N))_i = v_i - \overline{v}.$$

Applying ADMM and simplifying yields the following algorithm:

$$\begin{array}{rcl} x_i^{k+1} & := & \mathbf{prox}_{\lambda f_i}(x_i^k - \overline{x}^k - u^k) \\ u^{k+1} & := & u^k + \overline{x}^{k+1}, \end{array} \tag{5.12}$$

called the *proximal exchange algorithm*. The x_i-updates can be carried out independently in parallel as in the proximal consensus algorithm. When the exchange problem is feasible, the variables x_i^k converge to optimal values, and λu^k converges to a set of optimal commodity prices.

The proximal exchange algorithm has some interesting properties and interpretations. First, each agent does not need to know anything about the other agents; she only needs to know the current average market imbalance $\overline{x}^k$ and the current (scaled) price vector u^k. Each agent simply minimizes her own cost plus a quadratic regularization term that accounts for the current prices. This term is centered at $\overline{x}^k$, which are commodity quantities that would clear the market.

We note that the exchange and consensus problems can be viewed as duals. In particular, the constraint sets in the two problems are subspaces that are orthogonal complements of each other. If

$$A = \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix},$$

then the consensus set is the range of A and the equilibrium set is the null space of A^T.

5.3.2 General form exchange

Consider a problem setup just as in the general form consensus case, except that the constraint set is defined as

$$\mathcal{C} = \left\{ (z_1, \dots, z_N) \;\middle|\; \sum_{i\,:\,k \in c_i} (z_i)_k = 0 \right\}.$$

In other words, as before, each x_i is associated with one of N agents, and each of the n components corresponds to a different good or commodity. Here, there is a distinct market or exchange for each commodity j, so the equilibrium constraints only involve the participating agents. If each $c_j = [N]$, then we recover the global exchange formulation.

This can be visualized as Figure 5.3. The resulting algorithm is the same as in the global exchange case, except that only components that participate in given exchanges need to participate in de-meaning, and the mean is only computed across the subset of agents that appear in the constraint.

This exchange problem arises in the problem of dynamic energy exchange among (*e.g.*, ‘smart grid’) devices connected in a network [118].

Here, the commodities represent electrical energy at a specific node in a network in a specific time period. The market clearing constraints require that energy flow must balance at each energy exchange node in each time period. The agent objective terms include constraints on generation and consumption as well as a cost function.

5.4 Allocation

The *allocation problem* is given by

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N f_i(x_i) \\ \text{subject to} & x_i \geq 0, \quad i = 1, \ldots, N, \\ & \sum_{i=1}^N x_i = b, \end{array} \tag{5.13}$$

with variables $x_i \in \mathbf{R}^n$, $i = 1, \ldots, N$. This problem can be interpreted much like the exchange problem. There are n types of resources, each of which is to be allocated across N activities to minimize each activity cost f_i. There is a fixed amount b_j of each resource available, which justifies the nonnegativity and budget constraints.

As before, this can be written in the canonical form (5.1) as

$$\text{minimize} \quad \sum_{i=1}^N f_i(x_i) + I_{\mathcal{C}}(x_1, \ldots, x_N),$$

where $\mathcal{C}$ is the *allocation set*

$$\mathcal{C} = \{(x_1, \ldots, x_N) \mid x_i \geq 0, \ x_1 + \cdots + x_N = b\}.$$

The separability of f and g, and the partitions of $\mathcal{P}$ and $\mathcal{Q}$, are the same as in the previous two examples.

The resulting algorithm is

$$\begin{array}{rcl} x_i^{k+1} & := & \mathbf{prox}_{\lambda f_i}(x_i^k - z^k - u^k) \\ z^{k+1} & := & \Pi_{\mathcal{C}}(x^{k+1} + u^k) \\ u_i^{k+1} & := & u_i^k + x_i^{k+1} - z^{k+1}. \end{array} \tag{5.14}$$

Projecting onto $\mathcal{C}$ involves n independent projections onto the probability simplex in $\mathbf{R}^N$, which can be done using the method in §6.2.5.

We could also consider a version of the allocation problem with an inequality constraint $\sum_{i=1}^N x_i \leq b$, which does not require that the full budget of each resource be used. The discussion and algorithm above would be the same but with a slightly different projection operator.

5.5 Notes and references

Distributed optimization is a classic topic in optimization with a huge literature. Some classic and important modern references include those by Dantzig and Wolfe [68], Benders [23], Lasdon [120], Geoffrion [95], Tsitsiklis [191], Bertsekas and Tsitsklis [27], Censor and Zenios [51], and Nediȅ and Ozdaglar [146, 147]. Exchange and allocation problems are classical in the economics literature; see, *e.g.*, [193, 192, 1]. See the recent monograph by Boyd et al. [33] for additional discussion on many of the topics discussed above, particularly focused on large-scale applications in statistics and machine learning. We also mention that the 'divide-and-concur' method for (nonconvex) constraint satisfaction problems can be derived as a special case of a message-passing version of ADMM [74].

6

Evaluating Proximal Operators

We now discuss how to evaluate the proximal operator of a wide variety of functions. By definition, evaluating a proximal operator involves solving a convex optimization problem, so the simplest approach is to use a generic optimization algorithm, exploiting generic structure in the problem like sparsity. In many cases there is a simpler or faster specialized method, or even an analytical solution, for the problem. We turn to a variety of such examples after discussing the use of generic methods in more detail, but we emphasize that proximal methods can be very useful even in cases when a closed form solution for the proximal operator is not available.

When f is the indicator function of a set $\mathcal{C}$, the proximal operator is projection onto $\mathcal{C}$. Just as there is a close connection between a function and its epigraph, there is often a close connection between certain proximal operators and certain projection operators. For this reason, we do not separate a discussion of projection operators; instead, we try to group together operators that are conceptually related.

It is also important to keep in mind that the examples discussed in this chapter can be combined or extended in various ways by applying the properties from Chapter 2; we will see that the Moreau decomposition will be particularly useful. For another example, given the product set $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_k$ and the vector $v = (v_1, \ldots, v_k)$, partitioned conformably, we have that $(\Pi_{\mathcal{C}}(v))_i = \Pi_{\mathcal{C}_i}(v_i)$, *i.e.*, each component can be projected in parallel. This follows from applying the separable sum property from §2.1 to $I_{\mathcal{C}} = I_{\mathcal{C}_1} + \cdots + I_{\mathcal{C}_k}$. In many cases, we will also see that there are several different ways to derive the proximal operator of a particular function.

6.1 Generic methods

In general, the problem we wish to solve is

$$\begin{array}{ll} \text{minimize} & f(x) + (1/2\lambda)\|x - v\|_2^2 \\ \text{subject to} & x \in \mathcal{C}, \end{array} \qquad (6.1)$$

with variable $x \in \mathbf{R}^n$, where $\mathcal{C} = \mathbf{dom}\, f$ (which may be all of $\mathbf{R}^n$, in which case the problem is unconstrained).

If the problem is unconstrained, so $\mathcal{C} = \mathbf{R}^n$, then the properties of f determine which algorithms are applicable. For example, if f is a generic nonsmooth function, then we can use a subgradient method to solve the problem. If f is smooth, we can use a gradient method, Newton's method, quasi-Newton methods like L-BFGS, and so on. If the problem is constrained, we can use, for example, a projected subgradient method if f is nonsmooth and a projected gradient method or interior-point method if f is smooth. These and many other methods are discussed in, *e.g.*, Nocedal and Wright [155].

If (6.1) is representable in a canonical form, then we may transform it into such a canonical form and then use an off-the-shelf implementation of a solver for such problems. For example, if (6.1) is SDP-representable, then a parser-solver like CVX [100] could take a high-level description of the problem, transform it to a cone program, and solve it with a generic interior-point method-based cone solver.

6.1.1 Quadratic functions

If $f(x) = (1/2)x^T Ax + b^T x + c$, with $A \in \mathbf{S}^n_+$, then

$$\mathbf{prox}_{\lambda f}(v) = (I + \lambda A)^{-1}(v - \lambda b).$$

There are several important special cases of this result. For example, if $f(x) = b^T x + c$, *i.e.*, if f is affine, then $\mathbf{prox}_{\lambda f}(v) = v - \lambda b$. If $f(x) = c$, so f is a constant function, then $\mathbf{prox}_{\lambda f}(v) = v$, so the proximal operator is the identity. Finally, if $f = (1/2)\|\cdot\|_2^2$, then

$$\mathbf{prox}_{\lambda f}(v) = \left(\frac{1}{1+\lambda}\right) v,$$

sometimes called a *shrinkage operator*.

Evaluating the proximal operator of a quadratic involves, in general, solving a system of linear equations with coefficient matrix $I + \lambda A$:

$$(I + \lambda A)x = v - \lambda b.$$

There are a number of ways to carry this out much more quickly than in a naïve implementation, which would take $O(n^3)$ flops for each evaluation. The most basic is to exploit structure in A. If A is, for instance, tridiagonal, then the system can be solved in $O(n)$ flops. There are additional techniques that apply when evaluating the proximal operator repeatedly for different values of v (but with λ fixed).

If the linear system is solved with a direct method, then we can compute a factorization of the coefficient matrix $I + \lambda A$, cache this factorization, and then re-use this factorization in each subsequent evaluation of $\mathbf{prox}_f$ (which involves solving another linear system with coefficient matrix $I + \lambda A$). Depending on the structure of A, this can lead to substantial savings, since computing the factorization is typically more expensive than the subsequent back-solve. For example, when $I + \lambda A$ has no particular structure (*i.e.*, is treated as a dense matrix), the factorization cost is $O(n^3)$ and the cost of a subsequent solve is $O(n^2)$. This means that, after the first evaluation of $\mathbf{prox}_{\lambda f}$, we get a discount of a factor of n for subsequent evaluations.

If the system is solved with an iterative method, such as CG, then we can warm start each evaluation at the previous solution. This technique can also give substantial savings. See, *e.g.*, [33, §4.2–§4.3] for more discussion. Typically the number of iterations required to solve the linear system (to some given accuracy) drops to a small number as the overall proximal algorithm converges.

The comments above hold for the slightly more complex case of a convex quadratic function restricted to an affine set [33, §4.2.5]. Here too, evaluating the proximal operator reduces to solving a set of linear equations.

6.1.2 Smooth functions

As mentioned above, if f is a smooth function, we can use any number of standard methods to solve (6.1): a gradient method, a quasi-newton method like L-BFGS, conjugate gradient, and so on. Indeed, the quadratic regularization term will serve to help convergence, and because it only contributes entries to the diagonal of the Hessian, it does not impact any structure in the Hessian of f.

Here, we describe two ways in which we can improve upon simply using one of these standard methods. First, it is typically necessary to repeatedly solve (6.1) (*i.e.*, evaluate $\mathbf{prox}_f$) with different values of v. In this case, we can warm start the iterative method being used at the solution of the problem for the previous value of v or at v itself. This will often provide a very large speed improvement over solving the problem from scratch each time.

Second, we can exploit structure in the Hessian of f. This is directly analogous to the discussion in the previous section. For example, suppose f is twice continuously differentiable and that we use Newton's method to solve (6.1). The main effort in carrying out Newton's method is in solving the system $Hx = -g$ each iteration, where $H = \nabla^2 f(x)$ and $g = \nabla f(x)$. The standard approach to solving this (symmetric, positive definite) system would be to form the Cholesky factorization $H = LL^T$ and then solve the Newton system via $x = -L^{-T}L^{-1}g$ (forward and back substitution). Computing the Cholesky factorization costs $(1/3)n^3$ flops, which dominates this computation. (We ignore the cost of forming H and g here.)

Suppose H is the sum of a diagonal and a rank one matrix, *i.e.*,

$$H = D + zz^T,$$

where $D \in \mathbf{R}^{n\times n}$ is diagonal. By the matrix inversion lemma,

$$H^{-1} = D^{-1} - \frac{D^{-1}zz^TD^{-1}}{1 + z^TD^{-1}z},$$

so the Newton system can be solved in $O(n)$ rather than $O(n^3)$ flops.

This structure arises in the following manner. Suppose

$$f(x) = \gamma\left(\sum_{i=1}^n \psi_i(x_i) + b\right) + \sum_{i=1}^n \varphi_i(x_i).$$

For notational convenience, define z so $z_i = \psi_i(x_i)$ and define φ and ψ so $(\varphi(x))_i = \varphi_i(x_i)$ and $(\psi(x))_i = \psi_i(x_i)$. Then the gradient is

$$\nabla f(x) = \gamma'(y)\nabla\psi(x) + \nabla\varphi(x),$$

and the Hessian is

$$\nabla^2 f(x) = \gamma''(y)\nabla\psi(x)\nabla\psi(x)^T + \gamma'(y)\,\mathbf{diag}(\tilde{z}) + \mathbf{diag}(\nabla\varphi(x)),$$

where $\tilde{z}_i = \psi_i''(x_i)$ and $y = \mathbf{1}^T z + b$. It is clear from inspection that this matrix is the sum of a diagonal and a rank one matrix.

For example, the 'log-sum-exp' function

$$f(x) = \log\left(\sum_{i=1}^n \exp x_i\right)$$

follows this form with $\gamma = \log$, $\psi_i = \exp$, $b = 0$, and $\varphi_i = 0$, so the expression for the Hessian simplifies to

$$\frac{1}{(\mathbf{1}^T z)^2}\left((\mathbf{1}^T z)\,\mathbf{diag}(z) - zz^T\right),$$

where, as above, $z_i = \exp x_i$. The geometric mean of a set of numbers also takes this form.

More generally, if H is the sum of a diagonal $n \times n$ matrix and a low rank matrix with rank p, we can solve the system in $O(np^2)$ flops using the method described in [34, §9.7.2]. This reference also discusses exploiting other types of structure, such as bandedness or sparsity.

To summarize, the main point is that if f is smooth, then we can evaluate $\mathbf{prox}_f$ using a general algorithm like Newton's method, and we may be able to carry it out very quickly by exploiting structure in the Hessian of f. In particular, using Newton's method and the technique above, we can evaluate the proximal operators of the log-sum-exp function or the geometric mean in $O(n)$ flops; this follows because each iteration costs $O(n)$ flops and Newton's method typically takes at most 10-20 iterations to converge in practice.

6.1.3 Scalar function

The separable sum property (see §2.1) implies that evaluating the proximal operator of a fully separable function reduces to evaluating the proximal operator of a scalar convex function $f : \mathbf{R} \to \mathbf{R} \cup \{+\infty\}$. In some cases, these can be analytically evaluated; in other words, these are functions for which we can analytically solve the optimality condition $\lambda f'(x) + x = v$ or $\lambda \partial f(x) + x \ni v$ for $x \in \mathbf{R}$.

For example, for $f(x) = -\log x$, we have

$$\mathbf{prox}_{\lambda f}(v) = \frac{v + \sqrt{v^2 + 4\lambda}}{2}. \tag{6.2}$$

This example will come up in §6.7.5. For a nonsmooth example, if $f(x) = |x|$, then we have that

$$\mathbf{prox}_{\lambda f}(v) = \begin{cases} v - \lambda & v \geq \lambda \\ 0 & |v| \leq \lambda \\ v + \lambda & v \leq -\lambda. \end{cases} \tag{6.3}$$

This operation is called *soft thresholding* and is discussed further in §6.5.2, which discusses the proximal operator of the ℓ_1 norm (a fully separable function).

Using Moreau decomposition, we can also evaluate the proximal operators of the conjugates of these functions.

6.1.4 General scalar function

Now we discuss generic methods for evaluating the proximal operator of a scalar function, which can then be applied elementwise to compute the proximal operator of any fully separable function.

Localization method. Suppose we only have a subgradient oracle for f, *i.e.*, suppose we can obtain a subgradient of f at any point in its domain. We can evaluate $\mathbf{prox}_{\lambda f}$ efficiently using a localization method (see [35, §4.1]) similar (but superior) to bisection.

We begin with the interval $[l, u] = \mathbf{dom}\, f$ (which can be $(-\infty, \infty)$). If v is outside this interval, we return the interval endpoint closest to

v. (In general, when $v \notin \mathbf{dom}\, f$, $\mathbf{prox}_{\lambda f}(v) = \Pi_{\mathbf{dom}\, f}(v)$.) Otherwise, the algorithm repeats the following until $u - l < \epsilon$, where $\epsilon > 0$ is a given tolerance:

1. In the first iteration, let $x = v$; otherwise, let $x = (l + u)/2$. Obtain $h \in \partial f(x)$, so
$$g = h + (1/\lambda)(x - v) \in \partial \varphi(x),$$
where φ is the full proximal objective.

2. Update the localization interval via
$$[l, u] := [l, u] \cap \begin{cases} [x - \lambda g, x] & g > 0 \\ [x, x - \lambda g] & g < 0. \end{cases}$$

The algorithm proceeds by obtaining a new upper and lower bound on $x^\star$ each iteration. Since the interval is reduced by at least 50% each iteration, it converges in at most $\lceil \log_2(L/\epsilon) \rceil$ iterations, where $2L$ is the length of the localization interval $[l, u]$ after the first iteration.

We now show how the bounds are obtained when $g > 0$; the other case is similar. If $g > 0$, then $\varphi(z) \geq \varphi(x) + g(z - x)$ for all z, so all $z > x$ are suboptimal, *i.e.*, x is an upper bound on $x^\star$. The point $z = x - \lambda g$ is a lower bound because each $g_z \in \partial \varphi(z)$ is nonpositive. To see this, let $h_z \in \partial f(z)$. Because $g > 0$ and $\lambda > 0$, we have $z < x$, which implies that $h_z \leq h$ because the subdifferential is monotone. Let
$$g_z = h_z + (1/\lambda)(z - v) \in \partial \varphi(z).$$
Then
$$h_z + (1/\lambda)(z - v) = h_z + (1/\lambda)\left(x - \lambda(h + (1/\lambda)(x - v)) - v\right),$$
and the righthand side is $h_z - h$, which is nonpositive. Thus z is a lower bound for $x^\star$.

Guarded Newton method. If f is twice continuously differentiable, we can use a guarded Newton method to find $x^\star$. The method starts with the initial interval $[l, u]$ obtained after the first iteration above, an initial value $x = (l + u)/2$, and a guard parameter value $\alpha \in [0, 1)$. The algorithm is the same as the localization method above, except that in step 1, we find the next query point via

$$x := \Pi_{\alpha[l,u]}(x - \varphi'(x)/\varphi''(x)),$$

i.e., we project the (pure) Newton update $x - \varphi'(x)/\varphi''(x)$ onto the interval $\alpha[l, u]$, where $\alpha[l, u]$ denotes the interval $[l, u]$ scaled by α about its center:

$$\alpha[l, u] := [(u + l)/2 - \alpha(u - l)/2, (u + l)/2 + \alpha(u - l)/2].$$

If $\alpha = 0$, the interval $\alpha[l, u]$ consists only of the midpoint of the next interval, so this algorithm reduces to the localization method above. Typically, the Newton method will converge in fewer iterations than the localization method, but each iteration will be more expensive due to the more complex computation of the subsequent query point.

6.2 Polyhedra

Here, we consider the case of projection onto a polyhedron; the same discussion applies to evaluating the proximal operator of a convex quadratic function restricted to a polyhedron. The polyhedron is given by a set of linear equalities and inequalities

$$\mathcal{C} = \{x \in \mathbf{R}^n \mid Ax = b,\ Cx \le d\},$$

where $A \in \mathbf{R}^{m\times n}$ and $C = \mathbf{R}^{p\times n}$. The projection problem is

$$\begin{array}{ll} \text{minimize} & (1/2)\|x - v\|_2^2 \\ \text{subject to} & Ax = b, \quad Cx \le d. \end{array} \tag{6.4}$$

This problem is a quadratic program; the speed with which this problem can be solved is determined by n, m, and p, as well as the structure of A and C. To compute the proximal operator of a convex quadratic function, restricted to $\mathcal{C}$, we simply add the quadratic objective to the one in the problem (6.4).

Here, we discuss some simple special cases, but note that there is a substantial literature on efficient projections onto various kinds of polyhedra. See, *e.g.*, Bauschke [7, §3.3.6] or Barman et al. [5] for applications with some more exotic examples.

6.2.1 Solution via duality

When m and p are both much smaller than n, it is much more efficient to solve (6.4) via the dual. This corresponds to a case where we want to project a high-dimensional point onto a polyhedron described by just a few equalities and inequalities.

The dual function of (6.4) is the concave quadratic

$$g(\nu, \eta) = -\frac{1}{2}\left\| \begin{bmatrix} A \\ C \end{bmatrix}^T \begin{bmatrix} \nu \\ \eta \end{bmatrix} \right\|_2^2 + \left(\begin{bmatrix} A \\ C \end{bmatrix} v - \begin{bmatrix} b \\ d \end{bmatrix} \right)^T \begin{bmatrix} \nu \\ \eta \end{bmatrix},$$

where $\nu \in \mathbf{R}^m$ and $\eta \in \mathbf{R}^p$ are dual variables. The dual problem is

$$\begin{array}{ll} \text{maximize} & g(\nu, \eta) \\ \text{subject to} & \eta \geq 0. \end{array}$$

This is a QP with $m+p$ variables, as opposed to n variables. We recover the solution of the problem (6.4) as

$$x^\star = v - A^T \lambda^\star - C^T \nu^\star, \tag{6.5}$$

where $\nu^\star$ and $\eta^\star$ are optimal points for the dual problem.

Gram matrix caching. Under the assumption that n is large but $m+p$ is modest (say, under 1000 or so), we show how to reduce the solution of the QP to one large computation that can be easily parallelized. We first compute the Gram matrix GG^T, where

$$G = \begin{bmatrix} A \\ C \end{bmatrix} \in \mathbf{R}^{(m+p)\times n},$$

and the vector Gv. These have dimensions $(m+p)\times(m+p)$ and $(m+p)$, respectively; in particular, they are small.

To compute the Gram matrix, we express it as the sum

$$GG^T = \sum_{i=1}^n g_i g_i^T,$$

where $g_i = (a_i, c_i) \in \mathbf{R}^{m+p}$. In other words, we compute outer products of vectors in $\mathbf{R}^{m+p}$ independently in parallel, then compute an elementwise sum of all these matrices (*e.g.*, via an all-reduce operation in a distributed implementation). This Gram matrix computation is done only *once*. If the matrices involved are dense, this step requires $n(m+p)^2$ flops; it is easily parallelized. If the matrices are sparse, the cost is much less.

Each time we need to evaluate the projection, we first evaluate Gv. When the matrices involved are dense, this costs $n(m+p)$ flops, a savings by a factor of $m+p$ over the Gram matrix calculation; moreover, this step, like the Gram matrix calculation, is easily solved in a parallel implementation via an all-reduce operation. We then solve the dual QP, expressed as

$$\begin{array}{ll} \text{maximize} & (\nu,\eta)^T(GG^T)(\nu,\eta) + (Gv)^T(\nu,\eta) \\ \text{subject to} & \eta \geq 0. \end{array}$$

This is a small QP, which can be quickly solved. Finally, reconstructing $x^\star$ from $\lambda^\star$ and $\nu^\star$ via (6.5) requires $n(m+p)$ flops (when A and C are dense); this step is trivially parallelizable.

This approach permits solving (6.4) for more or less arbitrarily large n, as long as $m+p$ is of modest size. In a serial implementation, the cost is $O(n)$, but this can be reduced by a factor k with k processors.

6.2.2 Affine set

An affine set is a special case of a polyhedron for which there is an analytical expression for the projection. Let $\mathcal{C} = \{x \in \mathbf{R}^n \mid Ax = b\}$, where $A \in \mathbf{R}^{m \times n}$. Then

$$\Pi_{\mathcal{C}}(v) = v - A^\dagger(Av - b),$$

where $A^\dagger$ is the Moore-Penrose pseudoinverse of A [14, §4]. For example, if $m < n$ and A has full rank, then this specializes to

$$\Pi_{\mathcal{C}}(v) = v - A^T(AA^T)^{-1}(Av - b).$$

As in the previous section, we can compute AA^T (which is the Gram matrix) once and cache its factorization. After this initial work, each projection costs one multiplication by A and one by A^T, and each of these is easily parallelized.

As a special case, projection onto the hyperplane $\mathcal{C} = \{x \mid a^T x = b\}$ is given by

$$\Pi_{\mathcal{C}}(v) = v + \left(\frac{b - a^T v}{\|a\|_2^2}\right) a.$$

6.2.3 Halfspace

If $\mathcal{C} = \{x \mid a^T x \leq b\}$ is a halfspace, then

$$\Pi_{\mathcal{C}}(v) = v - \frac{(a^T v - b)_+}{\|a\|_2^2} a,$$

where $(u)_+ = \max\{u, 0\}$. In other words, we first check whether the point is in the halfspace, and if not, we project onto the hyperplane defining the boundary of the halfspace.

6.2.4 Box

Projection onto a *box* or *hyper-rectangle* $\mathcal{C} = \{x \mid l \leq x \leq u\}$ also takes a simple form:

$$(\Pi_{\mathcal{C}}(v))_k = \begin{cases} l_k & v_k \leq l_k \\ v_k & l_k \leq v_k \leq u_k \\ u_k & v_k \geq u_k, \end{cases}$$

i.e., we threshold the values at the boundary of the box. Here, l and u may be $-\infty$ or $+\infty$, so the box need not be bounded. For example, for $\mathcal{C} = \mathbf{R}^n_+$, we have

$$\Pi_{\mathcal{C}}(v) = v_+,$$

where the positive part operator is taken elementwise. In other words, to project a vector v onto the nonnegative orthant, each negative component of v is replaced with zero. This is a special case of projection onto a box with $l = 0$ and $u = \infty$.

6.2.5 Simplex

In other cases, there are simple iterative methods available. For example, to project $v \in \mathbf{R}^N$ onto the *probability simplex*

$$\mathcal{C} = \{z \mid z \geq 0,\ \mathbf{1}^T z = 1\},$$

it follows from the optimality conditions that

$$\Pi_{\mathcal{C}}(v) = (v - \nu \mathbf{1})_+$$

for some $\nu \in \mathbf{R}$. We carry out bisection on ν to find the value for which $\mathbf{1}^T(v - \nu\mathbf{1})_+ = 1$, starting with the initial interval $[\max_i v_i - 1, \max_i v_i]$. The function $\mathbf{1}^T(v - \nu\mathbf{1})_+$ is piecewise linear, with breakpoints at the values $v_1, \ldots, v_n$, so once we have localized ν to be between two adjacent values, we can immediately compute the optimal value $\nu^\star$.

6.3 Cones

Let $\mathcal{K}$ be a proper cone with dual cone $\mathcal{K}^*$. The optimality conditions of the problem

$$\begin{array}{ll} \text{minimize} & \|x - v\|_2^2 \\ \text{subject to} & x \in \mathcal{K}, \end{array}$$

with variable x, are given by

$$x \in \mathcal{K}, \qquad v = x - \lambda, \qquad \lambda \in \mathcal{K}^*, \qquad \lambda^T x = 0,$$

where λ is the dual variable for the cone constraint. Thus, projecting a point v onto $\mathcal{K}$ decomposes it into the difference of two orthogonal vectors x and λ, such that x is nonnegative with respect to $\mathcal{K}$ and λ is nonnegative with respect to $\mathcal{K}^*$. (This is an instance of the Moreau decomposition; see §2.5.)

We can derive many useful properties from the conditions above. For example, if $v \in \mathcal{K}^*$, then $\Pi_{\mathcal{K}}(v) = 0$. Next, we give several important special cases in which explicit solutions are available.

6.3.1 Nonnegative orthant

For the cone $\mathcal{C} = \mathbf{R}^n_+$, we have that

$$\Pi_{\mathcal{C}}(v) = v_+,$$

where the nonnegative part operator $(\cdot)_+$ is taken elementwise. Thus, to project onto $\mathbf{R}^n_+$, we simply replace each negative component of v with zero. (This is projection onto a box with $l = 0$ and $u = \infty$.)

6.3.2 Second-order cone

The *second-order cone* $\mathcal{C} = \{(x,t) \in \mathbf{R}^{n+1} \mid \|x\|_2 \leq t\}$ is also known as the *quadratic cone* or the *Lorentz cone*. Projection onto it is given by

$$\Pi_{\mathcal{C}}(v,s) = \begin{cases} 0 & \|v\|_2 \leq -s \\ (v,s) & \|v\|_2 \leq s \\ (1/2)(1 + s/\|v\|_2)(v, \|v\|_2) & \|v\|_2 \geq |s|. \end{cases}$$

6.3.3 Positive semidefinite cone

For the cone $\mathcal{C} = \mathbf{S}^n_+$, we have that

$$\Pi_{\mathcal{C}}(V) = \sum_{i=1}^{n} (\lambda_i)_+ u_i u_i^T, \tag{6.6}$$

where $\sum_{i=1}^n \lambda_i u_i u_i^T$ is the eigenvalue decomposition of V. In other words, to project a symmetric matrix onto the positive semidefinite cone, we form its eigenvalue expansion and drop terms associated with negative eigenvalues.

6.3.4 Exponential cone

The *exponential cone* $K_{\text{exp}} \subset \mathbf{R}^3$ is given by

$$K_{\text{exp}} = \{(x,y,z) \mid y > 0,\ ye^{x/y} \leq z\} \cup \{(x,y,z) \mid x \leq 0,\ y = 0,\ z \geq 0\}.$$

Its dual cone is

$$K^*_{\text{exp}} = \{(u,v,w) \mid u < 0,\ -ue^{v/u} \leq ew\} \cup \{(0,v,w) \mid v \geq 0,\ w \geq 0\}.$$

We can compute the projection $v = (r,s,t)$ of $v_0 = (r_0, s_0, t_0) \in \mathbf{R}^3$ onto K_{exp} as follows:

1. If $v_0 \in K_{\text{exp}}$, then $v = v_0$.

2. If $-v_0 \in K^*_{\text{exp}}$, then $v = 0$.

3. If $r_0 < 0$ and $s_0 < 0$, then $v = (r_0, (s_0)_+, (t_0)_+)$.

4. Otherwise, v is the solution to

$$\begin{array}{ll} \text{minimize} & (1/2)\|v - v_0\|_2^2 \\ \text{subject to} & se^{r/s} = t, \quad s > 0. \end{array}$$

(The constraint $s > 0$ cannot be active at the optimal point; it is really the domain of the constraint function.) This optimization problem can be solved with a primal-dual Newton method in the following fashion. Let $f(v) = (1/2)\|v - v_0\|_2^2$ and let $g(v) = se^{r/s} - t$. Then computing the Newton step involves solving the system

$$\begin{bmatrix} \nabla^2 f(v) + \lambda \nabla^2 g(v) & \nabla g(v) \\ \nabla g(v)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta v \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(v) + \lambda \nabla g(v) \\ g(v) \end{bmatrix},$$

where $\lambda \in \mathbf{R}$ is a dual variable. This simplifies to

$$\begin{bmatrix} I + \lambda \nabla^2 g(v) & \nabla g(v) \\ \nabla g(v)^T & 0 \end{bmatrix} \begin{bmatrix} \Delta v \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} v - v_0 + \lambda \nabla g(v) \\ se^{r/s} - t \end{bmatrix},$$

where

$$\nabla g(v) = \begin{bmatrix} e^{r/s} \\ e^{r/s}(1 - r/s) \\ -1 \end{bmatrix}, \qquad \nabla^2 g(v) = e^{r/s} \begin{bmatrix} 1/s & -r/s^2 & 0 \\ -r/s^2 & r^2/s^3 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

When carrying out the backtracking line search, one should backtrack either if the usual backtracking condition holds or if $s + \Delta s < 0$; this enforces the constraint $s > 0$.

6.4 Pointwise maximum and supremum

6.4.1 Max function

Let $f(x) = \max_i x_i$. In epigraph form, we have the equivalent problem

$$\begin{array}{ll} \text{minimize} & t + (1/2\lambda)\|x - v\|_2^2 \\ \text{subject to} & x_i \le t, \quad i = 1, \ldots, n, \end{array}$$

with variables $x_1, \ldots, x_n \in \mathbf{R}$ and $t \in \mathbf{R}$. The Lagrangian is

$$L(x, t, \mu) = t + (1/2\lambda)\|x - v\|_2^2 + \mu^T(x - t\mathbf{1}),$$

with dual variable μ, and the optimality conditions are

$$x_i^\star \leq t^\star,\ \mu_i^\star \geq 0,\ \mu_i^\star(x_i^\star - t^\star) = 0,\ (1/\lambda)(x_i^\star - v_i) + \mu_i^\star = 0,\ \mathbf{1}^T\mu^\star = 1.$$

If $x_i^\star < t^\star$, then the third condition implies that $\mu_i^\star = 0$, and if $x_i^\star = t^\star$, the fourth implies that $\mu_i^\star = (1/\lambda)(v_i - t^\star)$. Because $\mu_i^\star \geq 0$, this gives

$$\mu_i^\star = (1/\lambda)(v_i - t^\star)_+.$$

Substituting for $\mu_i^\star$ in the fifth condition gives

$$\sum_{i=1}^n (1/\lambda)(v_i - t^\star)_+ = 1.$$

This equation can be solved for $t^\star$ by bisection using the initial interval $[\min_i v_i - (1/n), \max_i v_i]$. This is the same equation that must be solved to project a point on the probability simplex, and the comments there, such as the fact that that $t^\star$ can be computed exactly in a finite number of steps, hold here as well. Once we have $t^\star$, we recover the solution to the original problem via

$$x_i^\star = \min\{t^\star, v_i\}.$$

This follows by applying the third and fourth conditions.

Another approach is to note that the max function is the conjugate of the indicator function of the probability simplex and to then use Moreau decomposition. (This explains why the same type of equation arises in both cases.)

6.4.2 Support function

If $\mathcal{C}$ is a convex set, then the *support function* of $\mathcal{C}$ is given by

$$S_{\mathcal{C}}(x) = \sup_{y \in C} y^T x.$$

The conjugate of the support function is the indicator function of the convex set, so $(S_{\mathcal{C}})^* = I_{\mathcal{C}}$. By Moreau decomposition, it follows that

$$\mathbf{prox}_{\lambda S_{\mathcal{C}}}(v) = v - \lambda \Pi_{\mathcal{C}}(v/\lambda).$$

The following example is due to Vandenberghe [195]. Let

$$f(x) = x_{[1]} + x_{[2]} + \cdots + x_{[k]},$$

the sum of the k largest components of x. (It is well-known that f is a convex function.) The main observation is that f can be expressed as the support function of the convex set

$$\mathcal{C} = \{y \mid 0 \preceq y \preceq \mathbf{1},\ \mathbf{1}^T y = k\}.$$

It follows from the result above that the proximal operator of f can be easily evaluated via projection onto $\mathcal{C}$, which can in turn be carried out using a method like the one described in §6.2.

6.5 Norms and norm balls

If $f = \|\cdot\|$ is a (general) norm on $\mathbf{R}^n$, then $f^* = I_{\mathcal{B}}$, where $\mathcal{B}$ is the unit ball for the dual norm. (This result also follows from §6.4.2 via the observation that the support function of a unit norm ball is precisely the dual norm.) By Moreau decomposition, it follows that

$$\begin{aligned} \mathbf{prox}_{\lambda f}(v) &= v - \lambda \mathbf{prox}_{f^*/\lambda}(v/\lambda) && (6.7)\\ &= v - \lambda \Pi_{\mathcal{B}}(v/\lambda). && (6.8) \end{aligned}$$

Thus, there is a close connection both between the proximal operator of a norm and its dual norm as well as between proximal operators of norms and projection operators onto unit norm balls.

6.5.1 Euclidean norm

For example, let $f = \|\cdot\|_2$, the Euclidean norm in $\mathbf{R}^n$. It is intuitively obvious that we can project onto the Euclidean unit ball $\mathcal{B}$ as follows:

$$\Pi_{\mathcal{B}}(v) = \begin{cases} v/\|v\|_2 & \|v\|_2 > 1 \\ v & \|v\|_2 \le 1. \end{cases}$$

In other words, if the point is outside the ball, we simply scale it to have unit Euclidean norm. It then follows that

$$\mathbf{prox}_{\lambda f}(v) = (1 - \lambda/\|v\|_2)_+ v = \begin{cases} (1 - \lambda/\|v\|_2)v & \|v\|_2 \ge \lambda \\ 0 & \|v\|_2 < \lambda. \end{cases}$$

This operator is sometimes called *block soft thresholding*.

6.5.2 ℓ_1 and ℓ_∞ norms

Similarly, we know that the unit ball $\mathcal{B}$ of the ℓ_∞ norm is a box, so as discussed in §6.2, it is very easy to project onto:

$$(\Pi_{\mathcal{B}}(v))_i = \begin{cases} 1 & v_i > 1 \\ v_i & |v_i| \leq 1 \\ -1 & v_i < -1. \end{cases}$$

Since the ℓ_∞ norm is the dual norm of the ℓ_1 norm, this also tells us how to evaluate the proximal operator of $f = \|\cdot\|_1$, *i.e.*, via

$$(\mathbf{prox}_{\lambda f}(v))_i = \begin{cases} v_i - \lambda & v_i \geq \lambda \\ 0 & |v_i| \leq \lambda \\ v_i + \lambda & v_i \leq -\lambda. \end{cases}$$

This is known as the (elementwise) *soft thresholding* operator and can be expressed more compactly as

$$\mathbf{prox}_{\lambda f}(v) = (v - \lambda)_+ - (-v - \lambda)_+. \tag{6.9}$$

An alternate derivation was given in §6.1.3.

It is a little less straightforward to evaluate the proximal operator of the ℓ_∞ norm or, equivalently, to project onto the ℓ_1 ball. If $\mathcal{B}$ is the unit ℓ_1 ball, then the projection onto $\mathcal{B}$ is given by the soft thresholding operator above, except that λ is not given in advance: If $\|v\|_1 \leq 1$, then $\lambda = 0$, and otherwise, we need to compute λ as the solution of

$$\sum_{i=1}^{n} (|v_i| - \lambda)_+ = 1.$$

Alternatively, the proximal operator of the ℓ_∞ norm can be evaluated using a technique similar to the one in §6.4.1. In both cases, the main computational work involves solving an equation similar to the one above, often via bisection. Despite no closed form solution being available, this can be carried out very quickly. Other algorithms for projecting onto the ℓ_1 ball are discussed in [78].

See Bogdan et al. [30] for an example of a statistical application of a sorted, weighted variant of the basic ℓ_1 norm and some fast specialized methods for evaluating its proximal operator. This is also related to the example in §6.4.2.

6.5.3 Elastic net

These examples can also be combined or extended in various ways. For example, *elastic net regularization* is the function

$$f(x) = \|x\|_1 + (\gamma/2)\|x\|_2^2$$

where $\gamma > 0$, *i.e.*, a linear combination of an ℓ_1 penalty and a quadratic penalty [209]. This function has a simple proximal operator:

$$\mathbf{prox}_{\lambda f}(v) = \left(\frac{1}{1+\lambda\gamma}\right)\mathbf{prox}_{\lambda\|\cdot\|_1}(v),$$

i.e., soft thresholding followed by multiplicative shrinkage.

6.5.4 Sum of norms

Another important case is *sum-of-norms regularization.* Let

$$f(x) = \sum_{g\in\mathcal{G}} \|x_g\|_2,$$

where $\mathcal{G}$ is a partition of $[n]$. Then it is easy to see that

$$(\mathbf{prox}_{\lambda f}(v))_g = \left(1 - \frac{\lambda}{\|v_g\|_2}\right)_+ v_g$$

for all $g \in \mathcal{G}$. This function is sometimes known as a ℓ_1/ℓ_2 norm or as a *group lasso* penalty, and the corresponding proximal operator is sometimes called *block soft thresholding.*

It can also be useful to consider cases in which the groups in $\mathcal{G}$ can overlap, so $\mathcal{G}$ is a cover of $[n]$ rather than a partition. For instance, when $\mathcal{G}$ is *tree-structured*, meaning that either two groups $g,\ g' \in \mathcal{G}$ are disjoint or one is a subset of the other, the proximal operator can still be evaluated in linear time, as discussed in [111, 4].

6.5.5 Matrix norms

Finally, we mention that there are also efficient and, in some cases, closed-form expressions for proximal operators of matrix norms (and projections onto their unit balls). These examples are best discussed in the general setting of matrix functions covered in §6.7.

6.6 Sublevel set and epigraph

In this section, we consider the problem of projecting onto a sublevel set or the epigraph of a closed proper convex function. We show how these projection operators are related to the proximal operator of f (or a function closely related to f).

6.6.1 Sublevel set

The *t-sublevel set of f* is

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

which we assume is nonempty. We assume that $v \notin \mathcal{S}$, meaning that $f(v) > t$, where v is the point to be projected; otherwise, the projection is trivially $\Pi_{\mathcal{S}}(v) = v$. The projection onto $\mathcal{S}$ can be computed using standard methods. Here, we show how to express the projection onto $\mathcal{S}$ using the proximal operator of f.

The optimality conditions for the projection are

$$0 \in x - v + \lambda \partial f(x), \qquad f(x) = t, \qquad \lambda > 0.$$

The first condition says that $\Pi_{\mathcal{S}}(v) = \mathbf{prox}_{\lambda f}(v)$, where, by the second and third conditions, $\lambda > 0$ satisfies

$$f(\mathbf{prox}_{\lambda f}(v)) = t.$$

We can find λ by bisection since the lefthand side is decreasing in λ. In other words, we can project onto $\mathcal{S}$ by evaluating the proximal operator of f, but we first need to find the parameter λ of the proximal operator.

To see that $f(\mathbf{prox}_{\lambda f}(v))$ is decreasing in λ, suppose x_λ minimizes $\varphi(x) + \lambda\psi(x)$. Then

$$\begin{aligned} \varphi(x_\alpha) + \alpha\psi(x_\alpha) &\le \varphi(x_\beta) + \alpha\psi(x_\beta) \\ \varphi(x_\beta) + \beta\psi(x_\beta) &\le \varphi(x_\alpha) + \beta\psi(x_\alpha). \end{aligned}$$

Rearranging these inequalities and adding them together gives

$$0 \le (\alpha - \beta)(\psi(x_\beta) - \psi(x_\alpha)),$$

so if $\alpha \ge \beta$, then $\psi(x_\beta) \ge \psi(x_\alpha)$, so $\psi(x_\lambda)$ is non-increasing in λ; the result follows from taking $\varphi(x) = (1/2)\|x - v\|_2^2$ and $\psi(x) = f(x)$.

6.6.2 Epigraph

Other than using a general purpose solver to project onto the epigraph, we have two characterizations of the projection. First, in general,

$$\Pi_{\mathbf{epi}\, f}(v,s) = (x, f(x)),$$

where x is the unique solution of

$$v \in x + (f(x) - s)\partial f(x).$$

Depending on f, this inclusion may or may not be easy to solve.

A second characterization is that

$$\Pi_{\mathbf{epi}\, f}(v,s) = (x,t),$$

where

$$x = \mathbf{prox}_g(v), \quad t = \max\{f(x), s\},$$

and g is given by $g(v) = (f(v) - s)_+^2$.

6.7 Matrix functions

We discuss two types of matrix functions: entrywise functions and matrix functions that can be viewed as vector functions of the eigenvalues or singular values of the matrix. Both these cases use proximal operators of corresponding vector functions as a building block, and together they cover most matrix functions of interest.

6.7.1 Elementwise functions

The first case is trivial. An entrywise matrix function treats a matrix $A \in \mathbf{R}^{m\times n}$ as a vector in $\mathbf{R}^{mn}$ and then uses a corresponding vector function; the proximal operator is then the same as that of the vector function. For example, the entrywise ℓ_1 norm of a matrix A is

$$\|A\|_1 = \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|,$$

and its proximal operator is elementwise soft thresholding. The entrywise ℓ_2 norm (the Frobenius norm) is similarly easy to handle.

6.7.2 Orthogonally invariant functions

The second case is conceptually simple but requires introducing several definitions; these results are based on those of Lewis [127, 128, 130], with some results tracing back to Davis [70].

A function $F : \mathbf{R}^{m\times n} \to \mathbf{R}$ is *orthogonally invariant* if

$$F(VXU) = F(X)$$

for all $X \in \mathbf{R}^{m\times n}$, $U \in \mathbf{R}^{n\times n}$, and $V \in \mathbf{R}^{m\times m}$, where U and V are orthogonal matrices. This implies, for example, that

$$F(X) = F(\mathbf{diag}(\sigma_s(X))),$$

where the *singular value map* $\sigma_s : \mathbf{R}^{m\times n} \to \mathbf{R}^{\min\{m,n\}}$ is the function that takes a matrix in $\mathbf{R}^{m\times n}$ and returns a vector of its singular values in nonincreasing order.

It turns out that F is orthogonally invariant if and only if $F = f\circ\sigma_s$, where f is *absolutely symmetric*, meaning that $f(Qx) = f(x)$ for all $x \in \mathbf{R}^p$ and any *signed permutation matrix* Q, *i.e.*, a matrix in which each row and each column has exactly one nonzero entry in $\{-1, +1\}$. Many properties of F can be derived from the corresponding properties of f; this is sometimes known as the *transfer principle*. For example, F is convex if and only if f is convex. Moreover, the subdifferential of a convex orthogonally invariant function is given by

$$\partial F(X) = \{V\,\mathbf{diag}(\mu)U \mid \mu \in \partial f(\sigma_s(X))\},$$

where $X = V\,\mathbf{diag}(\sigma_s(X))U$ is the singular value decomposition of X. In other words, we compute a subgradient of f at the singular value vector of X and then left and right multiply by V and U, respectively. This implies that

$$\mathbf{prox}_{\lambda F}(A) = V\,\mathbf{diag}(\mathbf{prox}_{\lambda f}(\sigma_s(A)))U, \tag{6.10}$$

i.e., we can evaluate the proximal operator of F by carrying out a singular value decomposition of A and evaluating the proximal operator of the corresponding absolutely symmetric function f at $\sigma_s(A)$.

Very similar results hold for functions $F : \mathbf{S}^n \to \mathbf{R}$ of symmetric matrices satisfying $F(UXU^T) = F(X)$ for all X and all orthogonal U;

such functions are called *spectral functions*. For instance, each spectral function F can be represented as $f \circ \sigma$, where $f : \mathbf{R}^n \to \mathbf{R}$ is a symmetric function and the *spectral map* σ takes a symmetric matrix and returns a vector of its eigenvalues in nonincreasing order. We have that

$$\mathbf{prox}_{\lambda F}(A) = U\,\mathbf{diag}(\mathbf{prox}_{\lambda f}(\sigma(A)))U^T \tag{6.11}$$

for any convex spectral function F, where $A = U\,\mathbf{diag}(\sigma(A))U^T$ is the eigendecomposition of A.

A closely related issue is to consider projections onto *spectral sets* in $\mathbf{S}^n$, which have the form $\sigma^{-1}(S)$ for any symmetric set $S \subseteq \mathbf{R}^n$. If F is a spectral function, then $\mathbf{dom}\,F$ is a spectral set, and $\sigma^{-1}(S)$ is convex if and only if S is. Following (6.11), if $S \subseteq \mathbf{R}^n$ is a symmetric convex set, then projection onto $T = \sigma^{-1}(S)$ is given by

$$\Pi_T(A) = U\,\mathbf{diag}(\Pi_S(\sigma(A)))U^T. \tag{6.12}$$

In other words, the effort involves computing the eigendecomposition of the argument and projecting the spectrum onto $S \subseteq \mathbf{R}^n$.

6.7.3 Matrix norms

Evaluating the proximal operator of orthogonally invariant matrix norms is now straightforward. For example, the *Schatten p-norm* of $A \in \mathbf{R}^{m\times n}$ is simply $\|\sigma_s(A)\|_p$, the ℓ_p norm of its singular values. Special cases include the *trace norm* or *nuclear norm* ($p = 1$), the *Frobenius norm* ($p = 2$), and the *spectral norm* ($p = \infty$).

It is straightforward to evaluate the proximal operator of a Schatten p-norm by building on the previous discussion. For example, if F is the nuclear norm (the ℓ_1 norm of the singular values), then

$$\mathbf{prox}_{\lambda F}(A) = \sum_{i=1}^{n}(\sigma_i - \lambda)_+ u_i v_i^T, \tag{6.13}$$

where $A = \sum_{i=1}^n \sigma_i u_i v_i^T$ is the singular value decomposition of A. This operation is called *singular value thresholding* since we soft threshold the singular values rather than the entries.

Another kind of orthogonally invariant norm is the *Ky Fan k-norm*, which is the sum of the k largest singular values of a matrix. We can evaluate its proximal operator by combining (6.10) with §6.4.2.

6.7.4 Projections onto spectral sets

It is easy to see that $\mathbf{S}^n_+ = \sigma^{-1}(\mathbf{R}^n_+)$, which lets us extend results from $\mathbf{R}^n_+$ to $\mathbf{S}^n_+$. Thus, we can project $A \in \mathbf{S}^n$ onto $\mathbf{S}^n_+$ by projecting $\sigma(A)$ onto $\mathbf{R}^n_+$; this is exactly (6.6), which we discussed previously.

As another example, we can project onto the convex spectral set

$$T = \{X \succeq 0 \mid \mathbf{Tr}\, X = 1\},$$

where $X \succeq 0$ means that X is positive semidefinite, via a projection onto the probability simplex S, since $T = \sigma^{-1}(S)$.

Finally, the unit ball $\mathcal{B}$ of the spectral norm is precisely $\sigma^{-1}(\mathcal{B}')$, where $\mathcal{B}'$ is the unit ℓ_∞ ball in $\mathbf{R}^n$. This gives that

$$\Pi_{\mathcal{B}}(A) = \sum_{i=1}^{n} \max\{d_i, 1\} u_i u_i^T, \tag{6.14}$$

where $A = \sum_{i=1}^n d_i u_i u_i^T$ is the eigenvalue decomposition of A. Thus, we can project a matrix onto $\mathcal{B}$ by thresholding its eigenvalues to have (absolute) magnitude at most 1. (We can also relate results like (6.14) to results like (6.13) via (6.8), as in the vector case.)

6.7.5 Log barrier

Consider the spectral function $F(A) = -\log\det A$. In this case, the corresponding f is simply the usual log barrier function, given by

$$f(x) = -\sum_{i=1}^{n} \log x_i.$$

It is easy to see that this function is separable down to the component and that its proximal operator is given by

$$(\mathbf{prox}_{\lambda f}(v))_i = \frac{v_i + \sqrt{v_i^2 + 4\lambda}}{2} \tag{6.15}$$

for $i = 1, \ldots, n$ (see (6.2)). This implies that the proximal operator of F can be evaluated by applying (6.15) to the spectrum of A.

6.8 Notes and references

Many, though not all, of the examples in this chapter are well-known in the literature; see, *e.g.*, Vandenberghe [195], Bauschke [7, §3.3], Bauschke and Combettes [10, chapter 28], Combettes and Pesquet [63], Zarantonello [205], and Boyd and Vandenberghe [34].

Many of the results on norms, especially the ℓ_1 norm and variants, come from the statistical and machine learning literature. See, for example, references on the lasso [184], soft thresholding [76], group lasso [203], sum-of-norms regularization [158], the CAP family of penalties [208], ℓ_1 trend filtering [116], covariance selection [73], sparse recovery [44], basis pursuit [58], Huber loss [107], singular value thresholding [46], and sparse graph selection [140].

There is a vast literature on projection operators and proximal operators for more exotic sets and functions. For a few representative examples, see, *e.g.*, [101, 169, 148, 182, 13]. We also highlight the paper by Chiercia et al. [59], which explores the connection between the proximal operator of a function and the projection onto its epigraph.

The material on orthogonally invariant matrix functions, spectral functions, and spectral sets is less widely known, though the literature on unitarily invariant matrix norms is classical and traces back at least to von Neumann in the 1930s.

The results used on spectral sets and functions are closely related to a general *transfer principle*: various properties of functions or sets in $\mathbf{R}^n$ can be 'transferred' to corresponding properties of functions or sets in $\mathbf{S}^n$; see, *e.g.*, [70, 65, 66, 67, 129, 181]. For general background on this area, also see, *e.g.*, [197, 89, 127, 128, 130, 71].

7

Examples and Applications

In this chapter we illustrate the main ideas we have discussed with some simple examples. Each example starts with a practical problem expressed in its natural form. We then show how the problem can be re-formulated in a canonical form amenable to one or more proximal algorithms, including, in some cases, parallel or distributed algorithms.

All the experiments were run on a machine with one (quad-core) Intel Xeon E3-1270 3.4 GHz CPU and 16 GB RAM running Debian Linux. The examples were run with MATLAB version 7.10.0.499. The source code for all the numerical examples is online at our website.

7.1 Lasso

The lasso problem is

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

with variable $x \in \mathbf{R}^n$, where $A \in \mathbf{R}^{m\times n}$, and $\gamma > 0$. The problem can be interpreted as finding a sparse solution to a least squares or linear regression problem or, equivalently, as carrying out simultaneous variable selection and model fitting.

7.1.1 Proximal gradient method

We refer here only to the basic version of the method, but everything also applies to the accelerated version.

Consider the splitting

$$f(x) = (1/2)\|Ax - b\|_2^2, \qquad g(x) = \gamma\|x\|_1, \tag{7.1}$$

with gradient and proximal operator

$$\nabla f(x) = A^T(Ax - b), \qquad \mathbf{prox}_{\gamma g}(x) = S_\gamma(x),$$

where S_λ is the soft-thresholding operator (6.9). Evaluating $\nabla f(x)$ requires one matrix-vector multiply by A and one by A^T, plus a (negligible) vector addition. Evaluating the proximal operator of g is neglible. Thus, each iteration of the proximal gradient method requires one matrix-vector multiply by A, one matrix-vector multiply by A^T, and a few vector operations. The proximal gradient method for this problem is sometimes called ISTA (iterative shrinkage-thresholding algorithm), while the accelerated version is known as FISTA (fast ISTA) [17].

There are ways to improve the speed of the basic algorithm in a given implementation. For example, we can exploit parallelism or distributed computation by using a parallel matrix-vector multiplication; see, *e.g.*, [94, 72, 29]. (The vector operations are trivially parallelizable.)

In special cases, we can improve efficiency further. If $n \ll m$, we can precompute the Gram matrix $A^TA \in \mathbf{S}^n_+$ and the vector $A^Tb \in \mathbf{R}^n$. The original problem is then equivalent to the (smaller) lasso problem

$$\text{minimize} \quad (1/2)\|\tilde{A}x - \tilde{b}\|_2^2 + \gamma\|x\|_1,$$

where $\tilde{A} = (A^TA)^{1/2}$ and $\tilde{b} = A^Tb$. (The objectives in the two problems differ by a constant.) This problem is small and can be solved very quickly: when $n \ll m$, all the work is in computing the Gram matrix A^TA (and A^Tb), which is now done only once.

These computations are also parallelizable using an all-reduce method, since each can be expressed as a sum over the rows of A:

$$A^TA = \sum_{i=1}^m a_i a_i^T, \qquad A^Tb = \sum_{i=1}^m a_i^T b,$$

where a_i^T are the rows of A. This also means, for example, that they can be computed only keeping a single $a_i \in \mathbf{R}^n$ in working memory at a given time, so it is feasible to solve a lasso problem with extremely large m on a single machine, as long as n is modest.

Another common situation in which a further efficiency improvement is possible is when the lasso problem is to be solved for many values of γ. For example, we might solve the problem for 50 values of γ, log spaced on the interval $[0.01\gamma_{\max}, \gamma_{\max}]$, where $\gamma_{\max} = \|A^T b\|_\infty$ is the critical value of γ above which the solution is $x^\star = 0$.

A simple and effective method in this case is to compute the solutions in turn, starting with $\gamma = \gamma_{\max}$, and initializing the proximal gradient algorithm from the value of $x^\star$ found with the previous, slightly larger, value of γ. Starting an iterative algorithm from a solution of a nearby problem is called *warm starting*. The same idea works for other cases, such as when we add or delete rows and columns of A, corresponding to observing new training examples or measuring new features in a regression problem. Warm starting can thus allow the (accelerated) proximal gradient method to be used in an online or streaming setting.

7.1.2 ADMM

To apply ADMM, we use the same splitting (7.1). Since f is quadratic, evaluating its proximal operator involves solving a linear system, as discussed in §6.1.1. We can thus apply the previous tricks:

- If a direct method is used to solve the subproblem, we can use *factorization caching*. This does mean, however, that the parameter λ must be kept fixed.

- If an iterative method is used, we can warm start the method at the previous proximal gradient iterate. In addition, we can use a loose stopping tolerance in the early iterations and tighten the tolerance as we go along. This amounts to evaluating the proximal operator of f or g approximately. (This simple variation on the basic method can be shown to work.)

- If $n \ll m$, we can precompute $A^T A$ and $A^T b$ (possibly in parallel fashion) to reduce the size of the problem.

Table 7.1: Comparing algorithms for the lasso. The error columns give the absolute and relative errors of solutions $x^\star$ compared to the solution found by CVX.

Method	Iterations	Time (s)	$p^\star$	Error (abs)	Error (rel)
CVX	15	26.53	16.5822	—	—
Proximal gradient	127	0.72	16.5835	0.09	0.01
Accelerated	23	0.15	16.6006	0.26	0.04
ADMM	20	0.07	16.6011	0.18	0.03

7.1.3 Numerical examples

We consider a small, dense instance of the lasso problem where the feature matrix $A \in \mathbf{R}^{m\times n}$ has $m = 500$ examples and $n = 2500$ features and is *dense*. We compare solving this problem with the proximal gradient method, the accelerated proximal gradient method, ADMM, and CVX (*i.e.*, transforming to a symmetric cone program and solving with an interior-point method).

We generate the data as follows. We first choose $A_{ij} \sim \mathcal{N}(0, 1)$ and then normalize the columns to have unit ℓ_2 norm. A 'true' value $x^{\text{true}} \in \mathbf{R}^n$ is generated with around 100 nonzero entries, each sampled from an $\mathcal{N}(0, 1)$ distribution. The labels b are then computed as $b = Ax^{\text{true}} + v$, where $v \sim \mathcal{N}(0, 10^{-3}I)$, which corresponds to a signal-to-noise ratio $\|Ax^{\text{true}}\|_2^2/\|v\|_2^2$ of around 200.

We solve the problem with regularization parameter $\gamma = 0.1\gamma_{\max}$, where $\gamma_{\max} = \|A^Tb\|_\infty$ is the critical value of γ above which the solution of the lasso problem is $x = 0$. We set $\lambda = 1$ in all three methods and set the termination tolerance to $\epsilon = 10^{-4}$ in each method's stopping criterion. All variables were initialized to zero.

In ADMM, since A is fat ($m < n$), we apply the matrix inversion lemma to $(A^TA + (1/\lambda)I)^{-1}$ and instead factor the smaller matrix $I + \lambda AA^T$, which is then cached for subsequent x-updates.

Table 7.1 gives the iterations, total time, and final error for proximal gradient and ADMM. Figure 7.1 shows the objective value by iteration. (The objective values in ADMM can be below the optimal value since the iterates are not feasible, *i.e.*, $x^k \neq z^k$.) We refer the reader to [33, 160] for additional examples.

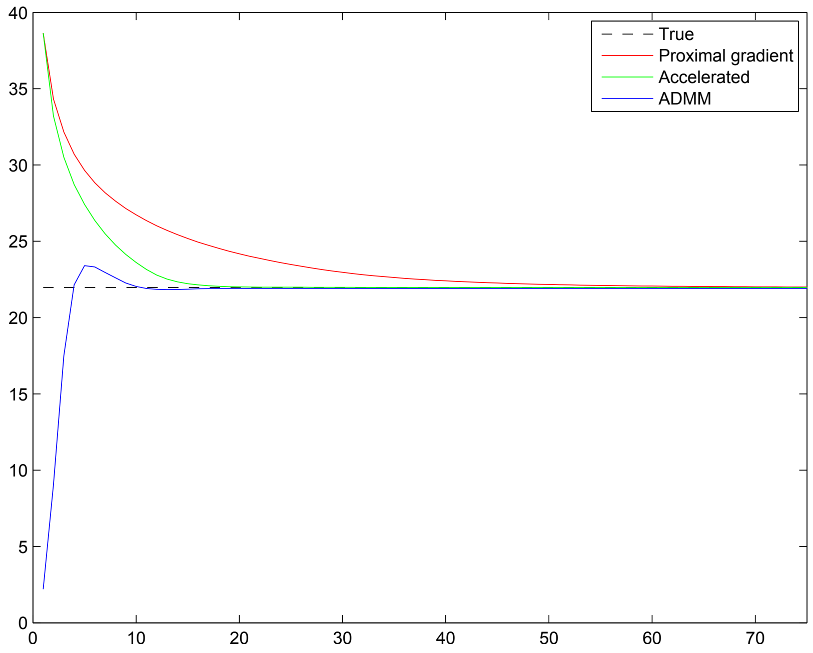

Figure 7.1: Objective values versus iteration for three proximal methods for a lasso problem. The dashed line gives the true optimal value. The ADMM objective values can be below the optimal value since the iterates are not feasible.

7.2 Matrix decomposition

A generic *matrix decomposition problem* has the form

$$\begin{array}{ll} \text{minimize} & \varphi_1(X_1) + \gamma_2\varphi_2(X_2) + \cdots + \gamma_N\varphi_N(X_N) \\ \text{subject to} & X_1 + X_2 + \cdots + X_N = A, \end{array} \tag{7.2}$$

with variables $X_1, \ldots, X_N \in \mathbf{R}^{m\times n}$, where $A \in \mathbf{R}^{m\times n}$ is a given data matrix and $\gamma_i > 0$ are trade-off parameters. The goal is to decompose a given matrix A into a sum of components X_i, each of which is 'small' or 'simple' in a sense described by the corresponding term φ_i. Problems of this type show up in a variety of applications and have attracted substantial recent interest; see, *e.g.*, [47, 54, 53, 133, 55, 178, 134].

We catalogue here some possibilities for the objective terms, which we also refer to as 'penalties':

- *Squared Frobenius norm.* $\varphi(X) = \|X\|_F^2 = \sum_{i,j} X_{ij}^2$. This penalty is a traditional least squares measure and encourages the entries in X to be small.

- *Entrywise ℓ_1 norm.* $\varphi(X) = \|X\|_1 = \sum_{i,j} |X_{ij}|$. This norm encourages X to be sparse (by serving as a convex surrogate for the number of nonzero entries in X).

- *Sum-column-norm.* $\varphi(X) = \sum_j \|x_j\|_2$, where x_j is jth column of X. This penalty encourages column sparsity in X, *i.e.*, choosing X with many zero columns. There is a corresponding row version. These can be interpreted as group lasso regularization (see §6.5.4) in which the groups are the rows or columns of the matrix.

- *Elementwise constraints.* We often want to constrain some or all entries to lie in some set, *i.e.*, $X_{ij} \in \mathcal{C}_{ij}$. We may even want to fix certain entries of the matrix to known values, to, *e.g.*, require X to be diagonal or to follow some fixed sparsity pattern. Another example is the box constraint $X_{ij} \in [l_{ij}, u_{ij}]$; a common special case is to require $X_{ij} \geq 0$.

- *Separable convex function.* Several of the previous items are simply examples of fully separable convex functions

$$\varphi(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \varphi_{ij}(X_{ij}),$$

  where $\varphi_{ij} : \mathbf{R} \to \mathbf{R} \cup \{+\infty\}$ is closed proper convex and may be extended-valued (*i.e.*, embed constraints). We could also consider functions of subblocks of the matrix, as in the case of the sum-column-norm and sum-row-norm.

- *Semidefinite cone constraint.* If X is a symmetric matrix, we may constrain $X \succeq 0$, so $\varphi(X) = I_{\mathbf{S}_+^n}(X)$.

- *Nuclear norm.* $\varphi(X) = \|X\|_*$ encourages X to be low rank (by serving as a convex surrogate for the nonconvex **rank** function). It can be viewed as the ℓ_1 norm of the singular values.

Decomposition problems involving sparse and low rank matrices have been of particular interest; see, *e.g.*, [54, 49]. For example, with

$$\varphi_1 = \|\cdot\|_F^2, \quad \varphi_2 = \|\cdot\|_1, \quad \varphi_3 = \|\cdot\|_*,$$

the problem (7.2) is to decompose a given matrix into a sum of a matrix with small entries, a sparse matrix, and a low rank matrix. Eliminating X_1 using $X_1 = A - (X_2 + X_3)$, the matrix decomposition problem is equivalent to the unconstrained problem

$$\text{minimize} \quad \|A - (X_2 + X_3)\|_F^2 + \gamma_2 \|X_2\|_1 + \gamma_3 \|X_3\|_*,$$

with variables X_2 and X_3. This matrix decomposition problem can be viewed as finding a good least squares approximation of A as a sum of a sparse (X_2) and a low rank (X_3) matrix. This problem is also closely related to *robust principal components analysis* [47]. For more discussion of the proximal operator of the nuclear norm, see, *e.g.*, [48].

7.2.1 ADMM algorithm

We give a generic method to solve (7.2) based on exchange (see §5.3) and ADMM. Consider the splitting

$$f(X) = \sum_{i=1}^{N} \varphi_i(X_i), \quad g(X) = I_{\mathcal{C}}(X),$$

where $X = (X_1, \dots, X_N)$ and

$$\mathcal{C} = \{(X_1, \dots, X_N) \mid X_1 + \cdots + X_N = A\}.$$

The algorithm requires two types of operations: evaluating the proximal operator of each φ_i and projecting onto $\mathcal{C}$. The latter set is similar to the equilibrium set (5.11) and so has a simple projection operator:

$$\Pi_{\mathcal{C}}(X_1, \dots, X_N) = (X_1, \dots, X_N) - \overline{X} + (1/N)A,$$

where $\overline{X}$ is the entrywise average of $X_1, \dots, X_N$. This can be obtained from the projection onto (5.11) (de-meaning) via (2.2): if $\mathcal{D} = z + \mathcal{C}$, where $\mathcal{C}$ and $\mathcal{D}$ are closed convex sets, then $\Pi_{\mathcal{D}}(v) = z + \Pi_{\mathcal{C}}(v - z)$.

Table 7.2: Comparing CVX and ADMM on a matrix decomposition problem.

Method	m	n	Iterations	Time (s)
CVX	10	30	15	1.11
ADMM	10	30	45	0.02
CVX	20	50	17	2.54
ADMM	20	50	42	0.03
CVX	40	80	20	108.14
ADMM	40	80	36	0.07
ADMM	100	200	38	0.58
ADMM	500	1000	42	35.56

The final algorithm looks as follows:

$$\begin{aligned} X_i^{k+1} &:= \mathbf{prox}_{\lambda\varphi_i}(X_i^k - \overline{X}^k + (1/N)A - U^k) \\ U^{k+1} &:= U^k + \overline{X}^{k+1} - (1/N)A, \end{aligned}$$

which is a minor variation on the exchange ADMM algorithm (5.12). Each iteration involves evaluating the proximal operator for each objective term (independently in parallel), plus some very simple entrywise matrix operations. Some of the proximal operators also involve entrywise operations (*e.g.*, soft thresholding) while some may require computing the singular value decomposition of the argument.

7.2.2 Numerical example

We consider the example problem described above for a few different sizes of m and n. We generate the data as follows. We chose $A = L + S + V$, where L is a rank 4 matrix, S is a sparse matrix, and V is a dense noise matrix. The matrix L is generated as $L = L_1 L_2$ with $L_1 \in \mathbf{R}^{m\times 4}$ and $L_2 \in \mathbf{R}^{4\times n}$, where entries in both L_1 and L_2 were sampled independently from $\mathcal{N}(0,1)$. The matrix S was generated with density 0.05, with each nonzero entry sampled uniformly from $\{-10, 10\}$. Each entry in V was sampled from $\mathcal{N}(0, 10^{-3})$.

We set $\gamma_2 = 0.15\gamma_2^{\max}$ and $\gamma_3 = 0.15\gamma_3^{\max}$, where $\gamma_2^{\max}$ is the entrywise ℓ_∞ norm of A and $\gamma_3^{\max}$ is the spectral norm of A. These values are the values above which the optimal values of X_2 and X_3, respectively, are zero. We take proximal parameter $\lambda = 1$.

Table 7.2 gives a summary of the computation required to solve the different problem instances. The larger instances were solved only with ADMM, since CVX would take too long or fail. Some comments:

- The problem sizes are nontrivial; *e.g.*, the 500×1000 instance has 1,500,000 variables and 500,000 equality constraints.
- In all the examples, the differences between the X_i found by ADMM and CVX were on the order of 0.01, measured in Frobenius norm. (The stopping tolerances could be tightened if needed; we used a modest tolerance of 10^{-4}, as in the previous example.)
- Though not relevant to simply solving the optimization problem, the solutions did well in the small instances and perfectly in the larger instances at estimating the rank of L and support of S. For example, the solution in the 40×80 example was perfect (rank 4 in L and X_3 and 156 nonzero entries in S and X_2).
- In ADMM, the final values of X_2 and X_3 are actually sparse and low rank, respectively, rather than only approximately so. By contrast, it is necessary to threshold entries and singular values of the X_2 and X_3 provided by CVX, respectively, after the fact.
- The cost of an ADMM iteration is effectively the cost of computing the (full) SVD of an $m \times n$ matrix. In addition, the number of iterations taken by ADMM does not necessarily increase in m and n. Thus, one could solve even some huge problems with 50 SVDs, at least to modest precision.

7.3 Multi-period portfolio optimization

Let $x_t \in \mathbf{R}^n$ denote the holdings of a portfolio of n assets at time period t, with $t = 1, \ldots, T$. The entry $(x_t)_i$ denotes the amount of asset i held in period t, considered as a real number, with a negative value indicating a short position. (These numbers can represent numbers of shares, dollar value of shares, or fractions of a total portfolio.) The initial portfolio x_0 is given. The goal is to find $x_1, \dots, x_T$ to minimize two costs: a risk-adjusted negative return and a transaction cost. We may also include additional constraints, such as a long-only constraint $x_t \geq 0$, requiring the final position to be zero ($x_T = 0$), or a normalization or 'budget' constraint $\mathbf{1}^T x_t = 1$, *e.g.*, when x_t represents the portfolio fractions invested in each asset.

The *multi-period portfolio optimization problem* is

$$\text{minimize} \quad \textstyle\sum_{t=1}^T f_t(x_t) + \sum_{t=1}^T g_t(x_t - x_{t-1}), \tag{7.3}$$

where f_t is the risk-adjusted negative return in period t and g_t is the transaction cost function. (The transaction cost term $g_t(x_t - x_{t-1})$ represents the cost of moving the portfolio to x_t from x_{t-1}.) We assume that f_t and g_t are closed proper convex and that g_t are fully separable, *i.e.*, the transaction cost in any period is the sum of the transaction costs for each asset. The variables in the multi-period portfolio problem (7.3) are the sequence of positions $x_1, \dots, x_T$; the data are x_0 and the functions f_t and g_t. We let $X = [x_1 \cdots x_T] \in \mathbf{R}^{n \times T}$ denote the matrix that gives the portfolio sequence.

Infinite values of the f_t and g_t impose constraints or limits on the portfolio holdings and the trades, respectively. The stage cost typically contains a quadratic risk term, a linear term that represents negative expected return, and constraints, such as a long-only constraint.

Consider the splitting

$$f(X) = \sum_{t=1}^T f_t(x_t), \qquad g(X) = \sum_{t=1}^T g_t(x_t - x_{t-1}),$$

i.e., we put the stage costs in one term and the transaction costs in the other. The function f is separable across the columns of X and the function g is separable across the rows of X (since the functions g_t are fully separable by assumption).

Evaluating the proximal operator of f is done by solving (in parallel) T single-period portfolio optimization problems

$$\text{minimize} \quad f_t(x_t) + (1/2\lambda)\|x_t - v_t\|_2^2,$$

for $t = 1, \ldots, T$. (The proximal regularization term can be viewed as an additional quadratic risk and expected return for each asset.)

The proximal operator of g can be found by solving n portfolio sequence problems in parallel, one for each asset i:

$$\text{minimize} \quad \textstyle\sum_{t=1}^{T} \left(g_{t,i}(x_{t,i} - x_{t-1,i}) + (1/2\lambda)\|x_{t,i} - v_{t,i}\|_2^2 \right),$$

with T variables $x_{1,i}, \ldots, x_{T,i}$. These problems can be interpreted as optimizing the schedule of buying and selling asset i, trading off transaction cost incurred by the buying and selling with matching a desired sequence of positions $v_{t,i}$. These sequence problems can be solved efficiently, in order $O(T)$ operations, by any method that can exploit the very specific structure that each variable $x_{t,i}$ appears in terms with only its time neighbors $x_{t+1,i}$ and $x_{t-1,i}$; see, *e.g.*, [34, §C.2].

7.3.1 Numerical example

We consider an example with stage cost

$$f_t(u) = -\mu^T u + (\gamma/2) u^T \Sigma u + I(u \geq 0) + I(\mathbf{1}^T x_t = 1), \quad t = 1, \ldots, T-1,$$

where the first term is the negative return, the second is the risk, and the third is a long-only constraint. The initial holding is $x_0 = 0$, and we require that the final position is zero, *i.e.*, $f_T(u) = I(u = 0)$. The transaction cost term is

$$g_t(u) = \kappa^T |u|^{3/2}, \quad t = 1, \ldots, T.$$

where the absolute value and 3/2-power are taken elementwise and $\kappa > 0$ is a vector of transaction cost coefficients. For simplicity, the stage cost and transaction cost functions do not depend on t (except for the final portfolio constraint).

We refer to the solution of the static allocation problem

$$\begin{array}{ll} \text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x \\ \text{subject to} & x \geq 0, \quad \mathbf{1}^T x = 1 \end{array}$$

as x^{static}, the optimal portfolio in the absence of transaction costs. The solution of the multiperiod portfolio problem will slowly build up a portfolio (so as to avoid excessive transcation costs), heading toward

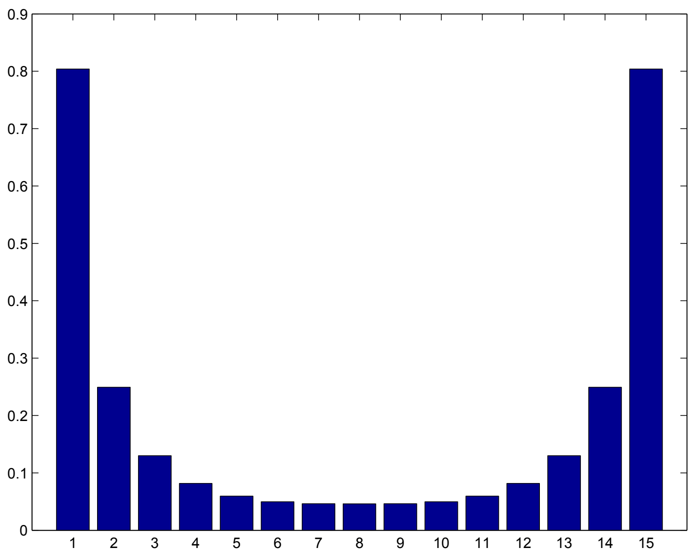

Figure 7.2: Time series of ℓ_1 deviation from x^{static}.

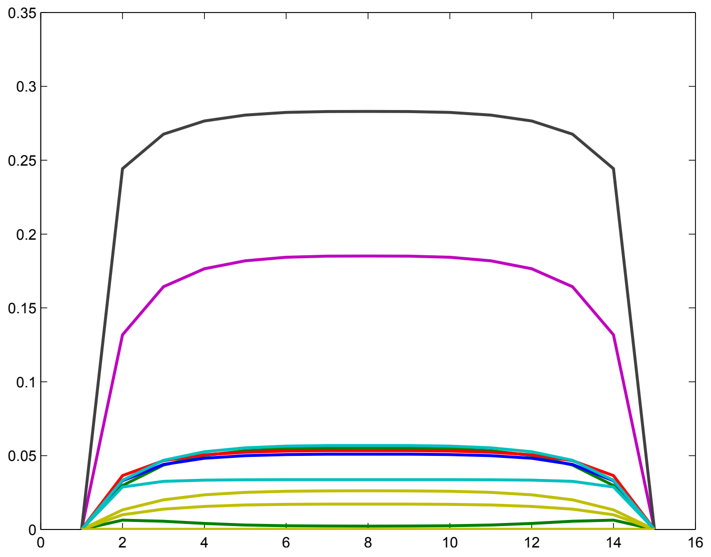

Figure 7.3: Time series of asset holdings.

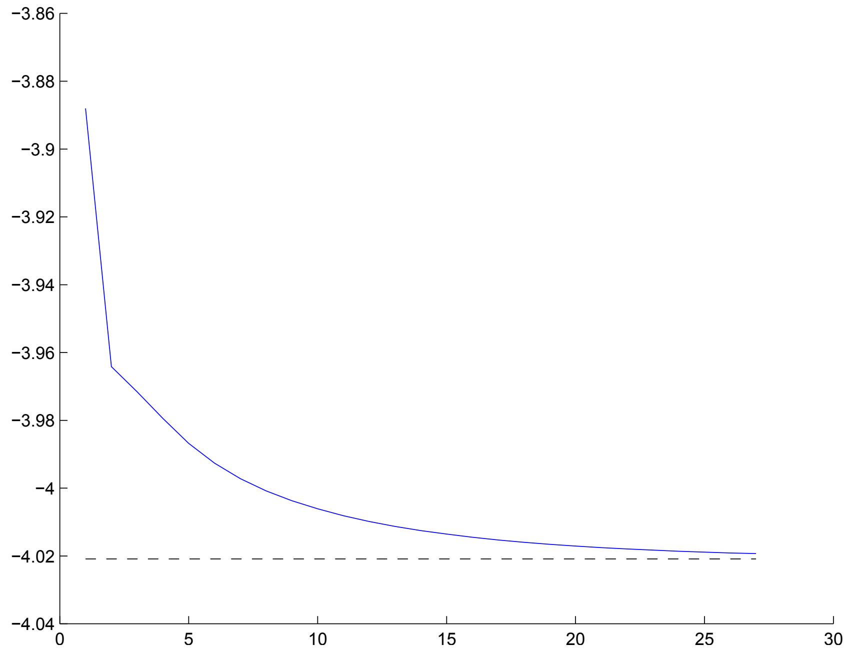

Figure 7.4: Convergence to optimality in finance example.

the optimal static portfolio, and near the end will sell its holdings over several periods (again, to avoid excessive transaction costs).

The instance we consider has $n = 20$ assets and $T = 15$ periods, for a total of 300 variables. The risk and return data are chosen randomly, and κ is chosen uniformly at random from $[0, 20]^n$. We use a risk aversion parameter of $\gamma = 1$ and use the ADMM parameter $\lambda = 1$.

Figure 7.2 shows $\|x_t^\star - x^{\text{static}}\|_1$ versus t. It shows that the portfolio builds up over first 4 or 5 periods, holds for around 5 periods within around 5% of x^{static}, and then liquidates over the remaining 5 periods. Figure 7.3 shows the optimal holdings $x_t^\star$ versus t. The solution invests most in assets with high positive returns (accounting for possibly higher risk levels) and does not invest in negative or low return assets. We see that in some cases, it quickly builds up a position in a certain asset and then reduces the position over time; this happens because it wishes to fully invest as soon as possible, and it chooses to enter positions that are relatively cheap to get into. It then reduces its positions in worse assets over time as it builds up larger positions in better assets.

This problem instance is very small and our implementation focused on readability, so we do not report detailed timing information; for reference, we show convergence to the optimal value in Figure 7.4. This problem instance, of course, could be easily solved on a single machine. We refer the reader to O'Donoghue et al. [157] for a much more detailed discussion of using ADMM for problems of this type. They also include experiments with highly tuned implementations and on very large problem instances. See Boyd et al. [32] for further discussion of multi-period investment problems.

7.4 Stochastic optimization

The *stochastic optimization problem* has the form

$$\text{minimize} \quad \textstyle\sum_{k=1}^{K} \pi_k f^{(k)}(x),$$

where $\pi \in \mathbf{R}_{+}^{K}$ is a probability distribution, *i.e.*, $\mathbf{1}^T\pi = 1$. The superscript k can be interpreted as indexing a *scenario*, so $f^{(k)}$ is the closed proper convex objective function for scenario k. Considering the scenario s as a discrete random variable with distribution π, the problem is to minimize $\mathbf{E}\, f^{(s)}(x)$, *i.e.*, to minimize the average objective value, where the average is taken over the various scenarios that could possibly occur.

This problem is simply that of minimizing an additive function. Following §5.2, we solve this problem by putting it into consensus form: We replace the variable x with new local variables $x^{(1)}, \dots, x^{(K)}$, replace the objective with the separable function

$$\sum_{k=1}^{K} \pi_k f^{(k)}(x^{(k)}),$$

and add a consensus constraint

$$x^{(1)} = \cdots = x^{(K)}.$$

We then apply the consensus ADMM method of §5.2. In each iteration, we evaluate the proximal operator of the objective for each scenario (independently in parallel), then average the local solutions $x^{(k)}$ for each scenario together.

7.5 Robust and risk-averse optimization

In *robust optimization*, we have K scenarios as in stochastic optimization, but instead of minimizing the expected value (average) of the objective, we minimize the maximum ('worst-case') objective:

$$\text{minimize} \quad \max_{k=1,\dots,K} f^{(k)}(x),$$

with variable $x \in \mathbf{R}^n$. This is also called a *minimax problem*.

A more general form of the problem is

$$\text{minimize} \quad \varphi\left(f^{(1)}(x), \dots, f^{(K)}(x)\right), \tag{7.4}$$

where φ is a convex nondecreasing function. This form includes stochastic optimization (with $\varphi(u) = \pi^T u$) and robust optimization (with $\varphi(u) = \max_k u_k$) as special cases. Another interesting case is

$$\varphi(u) = (1/\eta)\log\left(\pi_1 \exp \eta u_1 + \cdots + \pi_K \exp \eta u_K\right),$$

which gives *risk-averse optimization* [200], where $\eta > 0$ is a *risk aversion parameter* and the parameters π_k are probabilities. The name can be justified by the expansion of φ in the parameter η:

$$\varphi(u) = \mathbf{E}\, u + \eta\, \mathbf{var}(u) + o(\eta),$$

where $\mathbf{var}(u)$ is the variance of u (under the probabilities π_k).

7.5.1 Method

We turn to solving the general form problem (7.4). We put the problem in epigraph form, replicate the variable x and the epigraph variable t, and add consensus constraints, giving

$$\begin{array}{ll} \text{minimize} & \varphi(t^{(1)}, \dots, t^{(k)}) \\ \text{subject to} & f^{(k)}(x^{(k)}) \le t^{(k)}, \quad k = 1, \dots, K \\ & x^{(1)} = \cdots = x^{(k)}. \end{array}$$

This problem has (local) variables $x^{(1)}, \ldots, x^{(K)}$ and $t^{(1)}, \ldots, t^{(K)}$. We split the problem into two objective terms: the first is

$$\varphi(t^{(1)}, \ldots, t^{(k)}) + I_{\mathcal{C}}(x^{(1)}, \ldots, x^{(K)}),$$

where $\mathcal{C}$ is the consensus set, and the second is

$$\sum_{k=1}^{K} I_{\mathbf{epi}\, f^{(k)}}(x^{(k)}, t^{(k)}).$$

We refer to the first term as f and the second as g, as usual, and will use ADMM to solve the problem.

Evaluating the proximal operator of f splits into two parts that can be carried out independently in parallel. The first is evaluating $\mathbf{prox}_\varphi$ and the second is evaluating $\Pi_{\mathcal{C}}$ (*i.e.*, averaging). Evaluating $\mathbf{prox}_\varphi$ when φ is the max function (robust optimization) or log-sum-exp function (risk-averse optimization) is discussed in §6.4.1 and §6.1.2, respectively. Evaluating the proximal operator of g splits into K parts that can be evaluated independently in parallel, each of which involves projection onto $\mathbf{epi}\, f^{(k)}$ (see §6.6.2).

7.6 Stochastic control

In stochastic control, also known as optimization with recourse, the task is to make a sequence of decisions $x_0, \ldots, x_T \in \mathbf{R}^n$. In between successive decisions x_{t-1} and x_t, we observe a realization of a (discrete) random variable $\omega_t \in \Omega$; the ω_t are independent random variables with some known distribution on Ω. The decision x_t can depend on the realized values of $\omega_1, \ldots, \omega_t$ but not on the future values $\omega_{t+1}, \ldots, \omega_T$; the first decision x_0 is made without knowledge of any random outcomes. The constraints that reflect that decisions are made based only on what is known at the time are known as *causality constraints* or the *information pattern* constraint.

A *policy* $\varphi_1, \ldots, \varphi_T$ gives the decisions as a function of the outcomes on which they are allowed to depend:

$$x_t = \varphi_t(\omega_1, \ldots, \omega_t), \qquad \varphi_t : \Omega^{t-1} \to \mathbf{R}^n.$$

(We can think of $x_0 = \varphi_0$, where φ_0 is a function with no arguments, *i.e.*, a constant.) The policies φ_t are the variables in the stochastic control problem; they can be (finitely) represented by giving their values for all values of the random arguments. In other words, φ_t is represented by $|\Omega|^t$ vectors in $\mathbf{R}^n$. To specify the full policy, we must give

$$\sum_{t=0}^{T} |\Omega|^t = \frac{|\Omega|^{T+1} - 1}{|\Omega| - 1}$$

vectors in $\mathbf{R}^n$. (This is practical only for T small and $|\Omega|$ quite small, say, when the number above is no more than a thousand.)

The objective to be minimized is

$$\mathbf{E}\, \phi(x_0, \ldots, x_T, \omega_1, \ldots, \omega_T),$$

where $\phi : \mathbf{R}^{nT+1} \times \Omega^T$ is closed proper convex in its continuous arguments $x_0, \ldots, x_T$ for each value of the argument $\omega_1, \ldots, \omega_T$. The expectation is over $\omega_1, \ldots, \omega_T$; it is a finite sum with $|\Omega|^T$ terms, one for each outcome sequence. The stochastic control problem is to choose the policy that minimizes the objective.

The set of all possible outcome sequences, and the policy, is often shown as a T-depth $|\Omega|$-ary tree. This is shown in Figure 7.5 for a problem with possible outcomes $\Omega = \{a, b\}$ and $T = 3$ periods. Each node is labeled with the outcomes observed up to that point in time. The single vertex on the left corresponds to $t = 0$; the next two on the right correspond to $t = 1$. Each vertex gives a partial sequence of the outcomes; the leaves give a full sequence. A policy can be thought of as assigning a decision vector to each vertex of the tree. A path from the root to a leaf corresponds to a particular sequence of realized outcomes. The objective can be computed by summing the objective value associated with each path, multiplied by its probability.

7.6.1 Method

This problem can be expressed in consensus form by introducing a sequence of decision variables $x_0, \ldots, x_T$ for *each* of the $|\Omega|^T$ outcome sequences. We then impose the causality constraint by requiring that decisions at time t with the same outcomes up to time t be equal.

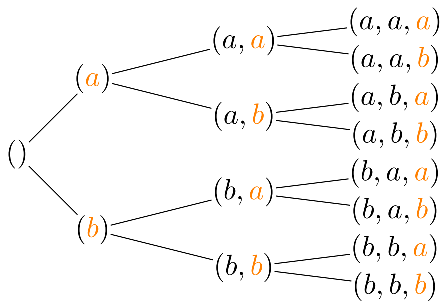

Figure 7.5: Tree of outcomes in stochastic control, with $|\Omega| = 2$ possible outcomes in $T = 3$ periods. A path from the root to a leaf corresponds to a full sequence of outcomes.

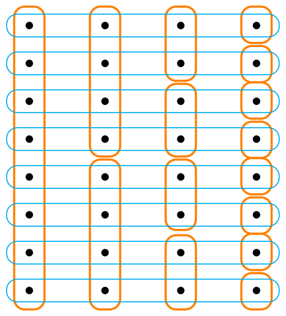

Figure 7.6: Stochastic control problem in consensus form. Each row corresponds to an outcome sequence. The cyan boxes show the separable structure of the objective. The orange boxes show the consensus constraints imposed by the causality requirement.

This is shown in Figure 7.6. Each row is a particular outcome sequence, so the objective (in blue) is separable across the rows of variables. The causality constraints (in orange) are consensus contraints: variables within each group must be equal. Ignoring the causality constraints gives the *prescient solution*, which is the best sequence of decisions if the full outcome sequence were known in advance.

Each iteration of ADMM has a natural interpretation and involves two main operations, corresponding to evaluation of the two proximal operators. Evaluating the proximal operator of the objective involves solving $|\Omega|^T$ independent optimization problems, one for each possible sequence of outcomes. Each of these subproblems finds something a bit like a prescient solution for a single outcome sequence (*i.e.*, a single row in Figure 7.6), taking into account a regularization term that prevents the solution from deviating too much from the previous consensus value (which does respect the causality constraints).

The second step involves a projection onto a consensus set, which enforces the causality constraints by averaging together the components of the prescient solutions where their corresponding outcome sequences agree (*i.e.*, each column of Figure 7.6). In other words, this averaged result z^k is a valid policy, so we consider the ADMM solution to be z^k rather than x^k at termination. We note that the consensus constraints for each orange group can be enforced independently in parallel. Eventually, we obtain decisions for each possible outcome sequence that are consistent with the causality constraints.

7.6.2 Numerical example

We consider a problem with $|\Omega| = 2$ equally probable outcomes over $T = 3$ periods with a state in $\mathbf{R}^{50}$. There are 8 possible outcome sequences, as shown in Figure 7.5, and we have an associated variable $x_\omega \in \mathbf{R}^{n(T+1)}$ for each of these outcomes. (We use ω to index the 8 outcome sequences.) Parts of these are constrained to be equal via the causality constraints. For example, the first half of x_{aaa} and x_{abb} must be the same: all the x_ω must agree on the first n values and x_{aaa} and x_{abb} must also agree on the second n values because they both correspond to a scenario that that observes outcome a in the first period.

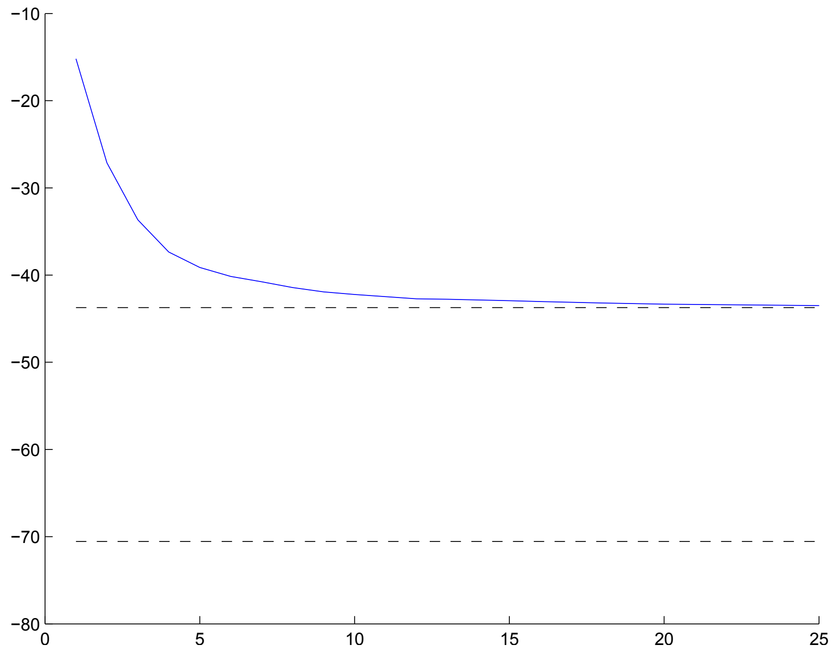

Figure 7.7: In blue, this shows the objective value attained by the iterate z^k in ADMM (which satisfies the causality constraints). The higher dashed black line is the optimal value of the full problem; the lower line is the optimal value attained by the prescient solution.

The objective functions for each outcome sequence are piecewise linear functions, with $m = 5$ components, plus the constraint that all components of every decision vector are in $[-1, 1]$:

$$\phi_\omega(u) = \max(A_\omega u + b) + I_{[-1,1]}(u),$$

where $A_\omega \in \mathbf{R}^{m \times 4n}$ and the max is over the m rows of the argument.

Figure 7.7 shows the objective value attained by the ADMM iterates z^k (which satisfy the causality constraints) progressing to the optimal value. The lower dashed line shows the (obviously superior) objective value attained by the prescient solution. Though ADMM takes about 50 iterations to terminate, we can see that we are very close to solving the problem within 20 iterations.

8 Conclusions

We have discussed proximal operators and proximal algorithms, and illustrated their applicability to standard and distributed convex optimization in general and many applications of recent interest in particular. Much like gradient descent and the conjugate gradient method are standard tools of great use when optimizing smooth functions serially, proximal algorithms should be viewed as an analogous tool for nonsmooth, constrained, and distributed versions of these problems.

Proximal methods sit at a higher level of abstraction than classical optimization algorithms like Newton's method. In such algorithms, the base operations are low-level, consisting of linear algebra operations and the computation of gradients and Hessians. In proximal algorithms, the base operations include solving small convex optimization problems (which in some cases can be done via a simple analytical formula).

Despite proximal algorithms first being developed nearly forty years ago, they are surprisingly well-suited both to many modern optimization problems, particularly those involving nonsmooth regularization terms, and modern computing systems and distributed computing frameworks. Many problems of substantial current interest in areas like machine learning, high-dimensional statistics, statistical signal processing, compressed sensing, and others are often more natural to solve using proximal algorithms rather than converting them to symmetric cone programs and using interior-point methods. Proximal operators and proximal algorithms thus comprise an important set of tools that we believe should be familiar to everyone working in such fields.

Acknowledgements

Neal Parikh was supported by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-0645962. This research was also supported by DARPA's XDATA program under Grant No. FA8750-12-2-0306.

We would like to thank Eric Chu, A. J. Friend, Thomas Lipp, Brendan O'Donoghue, Jaehyun Park, Alexandre Passos, Ernest Ryu, and Madeleine Udell for many helpful comments and suggestions. We are also very grateful to Heinz Bauschke, Amir Beck, Stephen Becker, Dmitriy Drusvyatskiy, Mário Figueiredo, Trevor Hastie, Adrian Lewis, Percy Liang, Lester Mackey, Ben Recht, Marc Teboulle, and Rob Tibshirani for reading earlier drafts and providing much useful feedback. We thank Heinz Bauschke, Amir Beck, Dmitriy Drusvyatskiy, Percy Liang, and Marc Teboulle in particular for very thorough readings.

References

[1] K.J. Arrow and G. Debreu. Existence of an equilibrium for a competitive economy. *Econometrica: Journal of the Econometric Society*, 22(3):265–290, 1954.

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

[3] H. Attouch. Convergence de fonctions convexes, de sous-différentiels et semi-groupes. *C.R Acad. Sci. Paris*, 284:539–542, 1977.

[4] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2011.

[5] S. Barman, X. Liu, S. Draper, and B. Recht. Decomposition methods for large scale LP decoding. In *Allerton Conference on Communication, Control, and Computing*, pages 253–260. IEEE, 2011.

[6] J. Barzilai and J.M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[7] H.H. Bauschke. *Projection Algorithms and Monotone Operators.* PhD thesis, Simon Fraser University, 1996.

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

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

[10] H.H. Bauschke and P.L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer-Verlag, 2011.

[11] H.H. Bauschke, P.L. Combettes, and D. Noll. Joint minimization with alternating Bregman proximity operators. *Pacific Journal on Mathematics*, 2:401–424, 2006.

[12] H.H. Bauschke, R. Goebel, Y. Lucet, and X. Wang. The proximal average: basic theory. *SIAM Journal on Optimization*, 19(2):766–785, 2008.

[13] H.H. Bauschke and V.R. Koch. Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces. See arXiv:1301.4506v1, 2013.

[14] H.H. Bauschke and S.G. Krug. Reflection-projection method for convex feasibility problems with an obtuse cone. *Journal of Optimization Theory and Applications*, 120(3):503–531, 2004.

[15] H.H. Bauschke, S.M. Moffat, and X. Wang. Firmly nonexpansive mappings and maximally monotone operators: correspondence and duality. *Set-Valued and Variational Analysis*, pages 1–23, 2012.

[16] A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

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

[18] A. Beck and M. Teboulle. Gradient-based algorithms with applications to signal recovery problems. In D.P. Palomar and Y.C. Eldar, editors, *Convex Optimization in Signal Processing and Communications*, pages 42–88. Cambribge University Press, 2010.

[19] A. Beck and M. Teboulle. Smoothing and first order methods: A unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

[20] S. Becker, J. Bobin, and E.J. Candès. NESTA: A fast and accurate first-order method for sparse recovery. *SIAM Journal on Imaging Sciences*, 4(1):1–39, 2011.

[21] S. Becker and M. Fadili. A quasi-Newton proximal splitting method. *Advances in Neural Information Processing Systems*, 2012.

[22] S.R. Becker, E.J. Candès, and M.C. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation*, pages 1–54, 2011.

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

[24] D.P. Bertsekas. Multiplier methods: A survey. *Automatica*, 12:133–145, 1976.

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

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

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

[28] J.M. Bioucas-Dias and M.A.T. Figueiredo. A new TwIST: two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16(12):2992–3004, 2007.

[29] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK user's guide*. SIAM: Philadelphia, 1997.

[30] M. Bogdan, E. van den Berg, W. Su, and E. Candès. Statistical estimation and testing via the sorted ℓ_1 norm. *arXiv:1310.1969*, 2013.

[31] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, and C.A. Sagastizábal. A family of variable metric proximal methods. *Mathematical Programming*, 68(1):15–47, 1995.

[32] S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment. To appear in *Foundations and Trends in Optimization*, 2013.

[33] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

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

[35] S. Boyd and L. Vandenberghe. Localization and cutting-plane methods. From Stanford EE 364b lecture notes, 2007.

[36] K. Bredies and D. Lorenz. Linear convergence of iterative soft-thresholding. *Journal of Fourier Analysis and Applications*, 14(5-6):813–837, 2008.

[37] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

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

[39] F.E. Browder. Multi-valued monotone nonlinear mappings and duality mappings in Banach spaces. *Transactions of the American Mathematical Society*, 118:338–351, 1965.

[40] F.E. Browder. Nonlinear monotone operators and convex sets in Banach spaces. *Bull. Amer. Math. Soc*, 71(5):780–785, 1965.

[41] F.E. Browder. Convergence theorems for sequences of nonlinear operators in Banach spaces. *Mathematische Zeitschrift*, 100(3):201–225, 1967.

[42] F.E. Browder. Nonlinear maximal monotone operators in Banach space. *Mathematische Annalen*, 175(2):89–113, 1968.

[43] R.D. Bruck. An iterative solution of a variational inequality for certain monotone operator in a Hilbert space. *Bulletin of the American Mathematical Society*, 81(5):890–892, 1975.

[44] A.M. Bruckstein, D.L. Donoho, and M. Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Review*, 51(1):34–81, 2009.

[45] R.S. Burachik and A.N. Iusem. *Set-Valued Mappings and Enlargements of Monotone Operators*. Springer, 2008.

[46] J.F. Cai, E.J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[47] E.J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? See arXiv:0912.3599, 2009.

[48] E.J. Candès, C.A. Sing-Long, and J.D. Trzasko. Unbiased risk estimates for singular value thresholding and spectral estimators. See arXiv:1210.4139, 2012.

[49] E.J. Candès and M. Soltanolkotabi. Discussion of 'latent variable graphical model selection via convex optimization'. *Annals of Statistics*, pages 1997–2004, 2012.

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

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

[52] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[53] V. Chandrasekaran, P.A. Parrilo, and A.S. Willsky. Latent variable graphical model selection via convex optimization. *Annals of Statistics* (with discussion), 2012.

[54] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, and A.S. Willsky. Sparse and low-rank matrix decompositions. In *Allerton 2009*, pages 962–967. IEEE, 2009.

[55] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, and A.S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011.

[56] G.H.G. Chen. *Forward-backward splitting techniques: theory and applications.* PhD thesis, University of Washington, 1994.

[57] G.H.G. Chen and R.T. Rockafellar. Convergence rates in forward-backward splitting. *SIAM Journal on Optimization*, 7(2):421–444, 1997.

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

[59] G. Chierchia, N. Pustelnik, J.-C. Pesquet, and B. Pesquet-Popescu. Epigraphical projection and proximal tools for solving constrained convex optimization problems: Part i. See arXiv:1210.5844, 2012.

[60] P. Combettes and J.-C. Pesquet. Proximal thresholding algorithm for minimization over orthonormal bases. *SIAM Journal on Optimization*, 18(4):1351–1376, 2007.

[61] P.L. Combettes. Solving monotone inclusions via compositions of non-expansive averaged operators. *Optimization*, 53(5-6), 2004.

[62] P.L. Combettes and J.-C. Pesquet. A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):564–574, 2007.

[63] P.L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212, 2011.

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

[65] A. Daniilidis, D. Drusvyatskiy, and A.S. Lewis. Orthogonal invariance and identifiability. arXiv:1304.1198, 2013.

[66] A. Daniilidis, A.S. Lewis, J. Malick, and H. Sendov. Prox-regularity of spectral functions and spectral sets. *Journal of Convex Analysis*, 15(3):547–560, 2008.

[67] A. Daniilidis, J. Malick, and H. Sendov. Locally symmetric submanifolds lift to spectral manifolds. arXiv:1212.3936, 2012.

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

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

[70] C. Davis. All convex invariant functions of Hermitian matrices. *Archiv der Mathematik*, 8(4):276–278, 1957.

[71] C.A. Deledalle, S. Vaiter, G. Peyré, J. Fadili, and C. Dossal. Risk estimation for matrix recovery with spectral regularization. See arXiv:1205.1482, 2012.

[72] J.W. Demmel, M.T. Heath, and H.A. Van Der Vorst. *Parallel numerical linear algebra*. Computer Science Division (EECS), University of California, 1993.

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

[74] N. Derbinsky, J. Bento, V. Elser, and J. Yedidia. An improved three-weight message-passing algorithm. *arXiv:1305.1961*, 2013.

[75] C.B. Do, Q.V. Le, and C.S. Foo. Proximal regularization for online and batch learning. In *International Conference on Machine Learning*, pages 257–264, 2009.

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

[77] J. Douglas and H.H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82:421–439, 1956.

[78] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine learning*, pages 272–279, 2008.

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

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

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

[82] E. Esser, X. Zhang, and T.F. Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[83] F. Facchinei and J.S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems.* Springer-Verlag, 2003.

[84] M.C. Ferris. Finite termination of the proximal point algorithm. *Mathematical Programming*, 50(1):359–366, 1991.

[85] M.A.T. Figueiredo, J.M. Bioucas-Dias, and R.D. Nowak. Majorization–minimization algorithms for wavelet-based image restoration. *IEEE Transactions on Image Processing*, 16(12):2980–2991, 2007.

[86] M.A.T. Figueiredo and R.D. Nowak. An EM algorithm for wavelet-based image restoration. *IEEE Transactions on Image Processing*, 12(8):906–916, 2003.

[87] M.A.T. Figueiredo and R.D. Nowak. A bound optimization approach to wavelet-based image deconvolution. In *IEEE International Conference on Image Processing*, volume 2, pages II–782. IEEE, 2005.

[88] G.F. Franklin, J.D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*, volume 3. Addison-Wesley: Reading, MA, 1994.

[89] S. Friedland. Convex spectral functions. *Linear and Multilinear Algebra*, 9(4):299–316, 1981.

[90] M. Fukushima and H. Mine. A generalized proximal point algorithm for certain non-convex minimization problems. *International Journal of Systems Science*, 12(8):989–1000, 1981.

[91] M. Fukushima and L. Qi. A globally and superlinearly convergent algorithm for nonsmooth convex minimization. *SIAM Journal on Optimization*, 6(4):1106–1120, 1996.

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

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

[94] K.A. Gallivan, R.J. Plemmons, and A.H. Sameh. Parallel algorithms for dense linear algebra computations. *SIAM Review*, pages 54–135, 1990.

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

[96] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 9:41–76, 1975.

[97] D. Goldfarb and K. Scheinberg. Fast first-order methods for composite convex optimization with line search. *preprint*, 2011.

[98] G.H. Golub and J.H. Wilkinson. Note on the iterative refinement of least squares solution. *Numerische Mathematik*, 9(2):139–148, 1966.

[99] A. Granas and J. Dugundji. *Fixed Point Theory*. Springer, 2003.

[100] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, ver. 1.1, build 630. Available at `www.stanford.edu/~boyd/cvx/`, April 2008.

[101] S.J. Grotzinger and C. Witzgall. Projections onto order simplexes. *Applied Mathematics and Optimization*, 12(1):247–270, 1984.

[102] O. Güler. On the convergence of the proximal point algorithm for convex minimization. *SIAM Journal on Control and Optimization*, 29:403, 1991.

[103] O. Güler. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2:649, 1992.

[104] E.T. Hale, W. Yin, and Y. Zhang. Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3):1107–1130, 2008.

[105] P.T. Harker and J.S. Pang. Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48(1):161–220, 1990.

[106] B. He and X. Yuan. On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

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

[108] D.R. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37, 2004.

[109] S. Ibaraki, M. Fukushima, and T. Ibaraki. Primal-dual proximal point algorithm for linearly constrained convex programming problems. *Computational Optimization and Applications*, 1(2):207–226, 1992.

[110] A.N. Iusem. Augmented Lagrangian methods and proximal point methods for convex optimization. *Investigación Operativa*, 8:11–49, 1999.

[111] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. See arXiv:1009.2139, 2010.

[112] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In *International Conference on Machine Learning*, 2010.

[113] R.I. Kachurovskii. Monotone operators and convex functionals. *Uspekhi Matematicheskikh Nauk*, 15(4):213–215, 1960.

[114] R.I. Kachurovskii. Non-linear monotone operators in Banach spaces. *Russian Mathematical Surveys*, 23(2):117–165, 1968.

[115] A. Kaplan and R. Tichatschke. Proximal point methods and nonconvex optimization. *Journal of Global Optimization*, 13(4):389–406, 1998.

[116] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky. ℓ_1 trend filtering. *SIAM Review*, 51(2):339–360, 2009.

[117] B.W. Kort and D.P. Bertsekas. Multiplier methods for convex programming. In *IEEE Conference on Decision and Control*, volume 12, 1973.

[118] M. Kraning, E. Chu, J. Lavaei, and S. Boyd. Message passing for dynamic network energy management. To appear, 2012.

[119] M. Kyono and M. Fukushima. Nonlinear proximal decomposition method for convex programming. *Journal of Optimization Theory and Applications*, 106(2):357–372, 2000.

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

[121] J. Lee, B. Recht, R. Salakhutdinov, N. Srebro, and J.A. Tropp. Practical large-scale optimization for max-norm regularization. *Advances in Neural Information Processing Systems*, 23:1297–1305, 2010.

[122] B. Lemaire. Coupling optimization methods and variational convergence. *Trends in Mathematical Optimization, International Series of Numerical Mathematics*, 84, 1988.

[123] B. Lemaire. The proximal algorithm. *International Series of Numerical Mathematics*, pages 73–87, 1989.

[124] C. Lemaréchal and C. Sagastizábal. Practical aspects of the Moreau-Yosida regularization I: theoretical properties, 1994. INRIA Technical Report 2250.

[125] C. Lemaréchal and C. Sagastizábal. Practical aspects of the Moreau–Yosida regularization: Theoretical preliminaries. *SIAM Journal on Optimization*, 7(2):367–385, 1997.

[126] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[127] A.S. Lewis. The convex analysis of unitarily invariant matrix functions. *Journal of Convex Analysis*, 2(1):173–183, 1995.

[128] A.S. Lewis. Convex analysis on the Hermitian matrices. *SIAM Journal on Optimization*, 6(1):164–177, 1996.

[129] A.S. Lewis. Derivatives of spectral functions. *Mathematics of Operations Research*, 21(3):576–588, 1996.

[130] A.S. Lewis and J. Malick. Alternating projections on manifolds. *Mathematics of Operations Research*, 33(1):216–234, 2008.

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

[132] F.J. Luque. Asymptotic convergence analysis of the proximal point algorithm. *SIAM Journal on Control and Optimization*, 22(2):277–293, 1984.

[133] S. Ma, L. Xue, and H. Zou. Alternating direction methods for latent variable gaussian graphical model selection. See arXiv:1206.1275, 2012.

[134] S. Ma, L. Xue, and H. Zou. Alternating direction methods for latent variable Gaussian graphical model selection. *Neural Computation*, pages 1–27, 2013.

[135] W.R. Mann. Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4(3):506–510, 1953.

[136] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[137] B. Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Revue Française de Informatique et Recherche Opérationelle*, 1970.

[138] B. Martinet. Détermination approchée d'un point fixe d'une application pseudo-contractante. *C.R. Acad. Sci. Paris*, 274A:163–165, 1972.

[139] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, pages 1–27, 2012.

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

[141] G.J. Minty. Monotone (nonlinear) operators in Hilbert space. *Duke Mathematical Journal*, 29(3):341–346, 1962.

[142] G.J. Minty. On the monotonicity of the gradient of a convex function. *Pacific J. Math*, 14(1):243–247, 1964.

[143] C.B. Moler. Iterative refinement in floating point. *Journal of the ACM (JACM)*, 14(2):316–321, 1967.

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

[145] J.-J. Moreau. Proximité et dualité dans un espace Hilbertien. *Bull. Soc. Math. France*, 93(2):273–299, 1965.

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

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

[148] A.B. Németh and S.Z. Németh. How to project onto an isotone projection cone. *Linear Algebra and its Applications*, 433(1):41–51, 2010.

[149] A.S. Nemirovsky and D.B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.

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

[151] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2004.

[152] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

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

[154] J.X.C. Neto, O.P. Ferreira, A.N. Iusem, and R.D.C. Monteiro. Dual convergence of the proximal point method with Bregman distances for linear programming. *Optimization Methods and Software*, pages 1–23, 2007.

[155] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[156] B. O'Donoghue and E. Candès. Adaptive restart for accelerated gradient schemes. See arXiv:1204.3982, 2012.

[157] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. To appear in *IEEE Transactions on Control Systems Technology*, 2012.

[158] H. Ohlsson, L. Ljung, and S. Boyd. Segmentation of ARX-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010.

[159] H. Ouyang, N. He, and A. Gray. Stochastic ADMM for nonsmooth optimization. See arXiv:1211.0632, 2012.

[160] N. Parikh and S. Boyd. Block splitting for distributed optimization. Submitted, 2012.

[161] G.B. Passty. Ergodic convergence to a zero of the sum of monotone operators in Hilbert space. *Journal of Mathematical Analysis and Applications*, 72(2):383–390, 1979.

[162] J.P. Penot. Proximal mappings. *Journal of Approximation Theory*, 94(2):203–221, 1998.

[163] R.A. Poliquin and R.T. Rockafellar. Prox-regular functions in variational analysis. *Transactions of the American Mathematical Society*, 348(5):1805–1838, 1996.

[164] B. Polyak. *Introduction to Optimization*. Optimization Software, Inc., 1987.

[165] B.T. Polyak. Iterative methods using Lagrange multipliers for solving extremal problems with constraints of the equation type. *USSR Computational Mathematics and Mathematical Physics*, 10(5):42–52, 1970.

[166] R. Polyak and M. Teboulle. Nonlinear rescaling and proximal-like methods in convex optimization. *Mathematical Programming*, 76(2):265–284, 1997.

[167] N. Pustelnik, C. Chaux, and J.-C. Pesquet. Parallel proximal algorithm for image restoration using hybrid regularization. *IEEE Transactions on Image Processing*, 20(9):2450–2462, 2011.

[168] N. Pustelnik, J.-C. Pesquet, and C. Chaux. Relaxing tight frame condition in parallel proximal methods for signal restoration. *IEEE Transactions on Signal Processing*, 60(2):968–973, 2012.

[169] A. Quattoni, X. Carreras, M. Collins, and T. Darrell. An efficient projection for $\ell_{1,\infty}$ regularization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 857–864, 2009.

[170] P. Ravikumar, A. Agarwal, and M.J. Wainwright. Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *Journal of Machine Learning Research*, 11:1043–1080, 2010.

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

[172] R.T. Rockafellar. On the maximal monotonicity of subdifferential mappings. *Pacific J. Math.*, 33(1):209–216, 1970.

[173] R.T. Rockafellar. On the maximality of sums of nonlinear monotone operators. *Transactions of the American Mathematical Society*, 149(1):75–88, 1970.

[174] R.T. Rockafellar. A dual approach to solving nonlinear programming problems by unconstrained optimization. *Mathematical Programming*, 5(1):354–373, 1973.

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

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

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

[178] J. Saunderson, V. Chandrasekaran, P.A. Parrilo, and A.S. Willsky. Diagonal and low-rank matrix decompositions, correlation matrices, and ellipsoid fitting. See arXiv:1204.1220, 2012.

[179] K. Scheinberg and D. Goldfarb. Fast first-order methods for composite convex optimization with large steps. Available online, 2012.

[180] K. Scheinberg, S. Ma, and D. Goldfarb. Sparse inverse covariance selection via alternating linearization methods. In *Advances in Neural Information Processing Systems*, 2010.

[181] H. Sendov. The higher-order derivatives of spectral functions. *Linear Algebra and its Applications*, 424(1):240–281, 2007.

[182] S. Sra. Fast projections onto $\ell_{1,q}$-norm balls for grouped feature selection. *Machine Learning and Knowledge Discovery in Databases*, pages 305–317, 2011.

[183] M. Teboulle. Entropic proximal mappings with applications to nonlinear programming. *Mathematics of Operations Research*, pages 670–690, 1992.

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

[185] K.C. Toh and S. Yun. An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems. *Preprint*, 2009.

[186] P. Tseng. Further applications of a splitting algorithm to decomposition in variational inequalities and convex programming. *Mathematical Programming*, 48(1):249–263, 1990.

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

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

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

[190] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *SIAM Journal on Optimization*, 2008.

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

[192] H. Uzawa. Market mechanisms and mathematical programming. *Econometrica: Journal of the Econometric Society*, 28(4):872–881, 1960.

[193] H. Uzawa. Walras' tâtonnement in the theory of exchange. *The Review of Economic Studies*, 27(3):182–194, 1960.

[194] L. Vandenberghe. Fast proximal gradient methods. From `http://www.ee.ucla.edu/~vandenbe/236C/lectures/fgrad.pdf`, 2010.

[195] L. Vandenberghe. Lecture on proximal gradient method. From `http://www.ee.ucla.edu/~vandenbe/shortcourses/dtu-10/lecture3.pdf`, 2010.

[196] L. Vandenberghe. Optimization methods for large-scale systems. UCLA EE 236C lecture notes, 2010.

[197] J. von Neumann. Some matrix inequalities and metrization of matrix space. *Tomsk University Review*, 1:286–300, 1937.

[198] J. von Neumann. *Functional Operators, Volume 2: The Geometry of Orthogonal Spaces*. Princeton University Press: Annals of Mathematics Studies, 1950. Reprint of 1933 lecture notes.

[199] Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented Lagrangian methods for semidefinite programming. Technical report, Department of IEOR, Columbia University, 2009.

[200] P. Whittle. *Risk-sensitive Optimal Control*. Wiley, 1990.

[201] S.J. Wright, R.D. Nowak, and M.A.T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

[202] K. Yosida. *Functional Analysis*. Springer, 1968.

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

[204] E.H. Zarantonello. *Solving functional equations by contractive averaging*. Mathematics Research Center, United States Army, University of Wisconsin, 1960.

[205] E.H. Zarantonello. Projections on convex sets in Hilbert space and spectral theory. I. Projections on convex sets. In *Contributions to Nonlinear Functional Analysis (Proceedings of a Symposium held at the Mathematics Research Center, University of Wisconsin, Madison, Wis., 1971)*, pages 237–341, 1971.

[206] X. Zhang, M. Burger, X. Bresson, and S. Osher. Bregmanized nonlocal regularization for deconvolution and sparse reconstruction. *SIAM Journal on Imaging Sciences*, 3(3):253–276, 2010.

[207] X. Zhang, M. Burger, and S. Osher. A unified primal-dual algorithm framework based on Bregman iteration. *Journal of Scientific Computing*, 46(1):20–46, 2011.

[208] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *Annals of Statistics*, 37(6A):3468–3497, 2009.

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