Chapter 8

Convex Optimization for Structured Signal Recovery

In the previous part, we showed rigorously that under fairly broad conditions, many important classes of structured signals can be recovered via computationally trackable optimization problems, such as ℓ^1 minimization for recovering sparse signals and nuclear norm minimization for recovering low-rank matrices. As we will see in Part III of this book, many of these structures are essential for modeling many high-dimensional data that arise in real-world applications. Hence, from a practical perspective, it is important that we develop efficient and scalable algorithms for these classes of optimization problems. We take on this task in this and the next couple of chapters.

In this chapter, we will mainly focus on the *convex approach* for structured signal recovery. There are two compelling reasons why we are in favor of studying the convex approach first. Firstly, the previous chapters have established very precise conditions under which convex programs give exactly correct solutions to the recovery problems, whereas conditions for other approaches are less strong or less understood. Secondly, as we will see through this chapter, the class of convex programs we are dealing with have very interesting and unique properties that lend themselves to much more fast and scalable solutions than generic convex programs. Although we will primarily use ℓ^1 norm or nuclear norm minimization as working examples, the techniques that we introduce are rather general. They can be easily applied to much larger class of convex programs that share the same properties and structures.

In the next two chapters, we will study two alternative approaches to develop algorithms for structured signal recovery. One approach is to rely

on simple and straightforward greedy strategies. This class of techniques are important because in many practical applications one may need to trade the range of working conditions of algorithms for better scalability or simplicity for implementation. Another approach is to explore stronger, not necessarily convex, regularizers for the desired structures such as sparsity. In practice, these regularizers are often observed to enjoy a wider range of working conditions than convex regularizers.

This chapter (or book) is not intended to give a comprehensive introduction to (convex) optimization[1]. Instead, the chapters will focus mainly on showing how one can exploit special structures of the problems so as to develop significantly more efficient and scalable algorithms than generic convex optimization methods. To make the book better self-contained, we will briefly survey related concepts and properties of convex functions as well as generic optimization methods in Appendices B-D.

8.1 Challenges and Opportunities

We have shown that under fairly broad conditions, many important classes of structured signals can be recovered by convex optimization – in particular, we analyzed in depth the ℓ^1 heuristic for recovering sparse signals and the nuclear norm heuristic for recovering low-rank matrices. As we will see later in the book, many of these structures are essential for modeling many high-dimensional data that arise in real-world applications. So, from a practical perspective, it is important to have efficient, scalable algorithms for these classes of optimization problems.

In this chapter, we will describe a few basic ideas which go a long way towards achieving this, by exploiting special properties of the particular convex optimization problems that arise in structured signal recovery. Our discussion will center around four model problems: basis pursuit (i.e., equality constrained ℓ^1 minimization), its penalized version (basis pursuit denoising), robust PCA, and its penalized version. We recap these four optimization problems below:

Recall the problem of recovering a sparse vector $\boldsymbol{x}_o \in \mathbb{R}^n$ from observations $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o \in \mathbb{R}^m$ via convex program, also known as *basis pursuit* (BP):

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{x}\|_1 \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array} \tag{8.1.1}$$

A variant of the problem considers the noisy case, in which the observations $\boldsymbol{y}$ are contaminated by moderate Gaussian noise $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o + \boldsymbol{z}$, also known

[1] for which there are already excellent references such as [Boyd and Vandenberghe, 2004].

as the *Lasso*:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1, \tag{8.1.2}$$

where λ is a scalar weight parameter.

In robust PCA, the goal is to recover a low-rank matrix $\boldsymbol{L}_o$ from sparsely corrupted observations $\boldsymbol{Y} = \boldsymbol{L}_o + \boldsymbol{S}_o$. A natural approach suggested in earlier chapters is to solve the so called *principal component pursuit* (PCP) program:

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{L}\|_* + \lambda \|\boldsymbol{S}\|_1 \\ \text{subject to} & \boldsymbol{L} + \boldsymbol{S} = \boldsymbol{Y}, \end{array} \tag{8.1.3}$$

where $\lambda > 0$ is a scalar weight. Again, if the data are noisy we could also consider to solve a stable version of the PCP program:

$$\text{minimize} \ \ \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{L} - \boldsymbol{S}\|_F^2 + \lambda_1 \|\boldsymbol{L}\|_* + \lambda_2 \|\boldsymbol{S}\|_1, \tag{8.1.4}$$

to produce stable estimates $\hat{\boldsymbol{L}}$ and $\hat{\boldsymbol{S}}$.

Challenge of scale.

When the dimension of the problem is not so high, one could simply apply classical second-order convex optimization algorithms, such as the interior-point methods (see e.g. [Boyd and Vandenberghe, 2004]), to solve the above convex programs. These require very few iterations to converge to a highly accurate solution: $O(\log\log(1/\varepsilon))$, where ε is the target accuracy. However, for problems in n variables, each iteration requires the solution of a linear system of equations of size $n \times n$, incurring a per-iteration cost of $O(n^3)$. For applications in modern signal processing, the number of variables can be quite high – in the case of image processing, it is typically of the same magnitude as the number of pixels, easily in the range of millions. For such problems, the cost of a single iteration is prohibitively large. We will need to consider simple alternatives with cheaper iterations.

Example 8.1.1 (Solving large-scale BP problems via interior-point methods)**.** *Just to motivate yourself from practical perspective, construct a simulation to plot the average run time of BP on CVX from 100 to 1000 dimensions. See how a generic convex optimization solver (that is mainly based on second-order, interior point method) scales for this class of optimization problems.*

Difficulty with non-smoothness.

The large scale of the problems forces us to consider simple, scalable algorithms that use only first-order information about the objective function. The prototypical first-order method is *gradient descent*. However, one technical difficulty arises though as the objective functions may contain non-smooth terms that are not differentiable. For instance, the ℓ^1 norm

$\|\boldsymbol{x}\|_1$ in the BPDN problem does not have a gradient in the normal sense. In such cases, the simplest solution is to employ a generic subgradient method, as we did in Chapter 2. Although the subgradient method has very simple and efficient iterations, its rate of convergence is very poor, typically $O(1/\sqrt{k})$.[2] That means it usually takes many (thousands of) iterations for the algorithm to converge to the optimal solution. In this chapter, we will show how to exploit some important properties of structured signal recovery. Such properties allow us to develop gradient descent algorithms as if the objective is smooth, the so called Proximal Gradient (PG) method. The same properties also allow us to utilize acceleration techniques that were designed for smooth functions and lead to much more scalable and fast-converging algorithms, with convergence rates much better than the generic situation.

Enforcing equality constraints.

To solve the basis pursuit problem (8.1.1), we need to ensure that the final solution $\boldsymbol{x}$ satisfies the equality constraint $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ exactly. A naive way to enforce the equality constraint is to incorporate it as a penalty term and minimize: $\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$. This is a similar optimization problem as the BPDN only that we need to solve a series of problems of this type for an increasing sequence of $\mu_i \to \infty$ so that the equality constraint is to be enforced in the end. However, as the weight μ_i increases, the corresponding BPDN problem becomes increasingly ill-conditioned and hence algorithms converge slower. In this chapter, we will see how to employ the *augmented Lagrangian multiplier* (ALM) technique to alleviate this difficulty.

Exploiting separable structures.

Very often the structured signal that we are recovering is a superposition of multiple structured terms. This is the case for the principal component pursuit (PCP) program that we have mentioned earlier:

$$\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 \quad \text{subj. to} \quad \boldsymbol{Y} = \boldsymbol{L} + \boldsymbol{S}. \tag{8.1.5}$$

As we will show in this chapter, such separable structures of the objective function can be naturally exploited through methods such as *alternating direction of multipliers method* (ADMM). We end up with more simple and efficient algorithms for solving this class of convex programs as ADMM converts the global optimization to several parallel subproblems of much smaller dimension.

[2] See [Nemirovski, 1995, Nemirovski, 2007] for a characterization of typical subgradient methods.

8.2 Proximal Gradient Methods

As one can see, the optimization problems we are dealing with have a common structure: they all try to minimize convex objective functions of the form:

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x}), \tag{8.2.1}$$

where $f(\boldsymbol{x})$ is a smooth convex term and $g(\boldsymbol{x})$ is convex but non-smooth term. For instance, in the Lasso problem (8.1.2), we could set $f(\boldsymbol{x}) \doteq \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ and $g(\boldsymbol{x}) \doteq \lambda\|\boldsymbol{x}\|_1$. We want to develop both scalable and fast-converging algorithms for this type of problems.

Since the composite objective function $F(\boldsymbol{x})$ is not differentiable, general gradient descent algorithms do not apply. The first recourse in this situation is to replace the gradient with a *subgradient*, yielding the simple subgradient descent iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma_k \boldsymbol{g}_k, \qquad \boldsymbol{g}_k \in \partial F(\boldsymbol{x}_k). \tag{8.2.2}$$

The main disadvantage to this approach is its relatively poor convergence rate. In general, the convergence rate of subgradient descent for non-smooth objective functions is (see [Nesterov, 2003b]):

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star) = O(1/\sqrt{k}). \tag{8.2.3}$$

The constants in the big-O notation depend on various properties of the problem. The important point is that for even a moderate target accuracy ε, we will have to set $k = \Omega(\varepsilon^{-2})$ very large.

Interlude: convergence rates of first order methods.

We can contrast the behavior of the subgradient method with the behavior of the simple *gradient descent* method for minimizing a smooth function. Consider, briefly, the simpler problem

$$\text{minimize} \quad f(\boldsymbol{x}), \tag{8.2.4}$$

with f convex and differentiable. The gradient descent iteration for this problem is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k). \tag{8.2.5}$$

This iteration comes from a first-order approximation to f at $\boldsymbol{x} = \boldsymbol{x}_k$:

$$f(\boldsymbol{x}') \;\geq\; f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle\,. \tag{8.2.6}$$

Because f is convex, this first order approximation provides a global lower bound on f. Nevertheless, we expect this lower bound to be more accurate in a neighborhood of $\boldsymbol{x}$. The size of this neighborhood depends substantially on the properties of f. For example, if f is relatively smooth, and its gradient does not vary much from point to point, we might imagine that the first order approximation at $\boldsymbol{x}$ would be accurate over a relatively large

region. To make this more formal, we say that a differentiable function $f(\boldsymbol{x})$ has *L-Lipschitz gradient* if

$$\|\nabla f(\boldsymbol{x}_1) - \nabla f(\boldsymbol{x}_2)\| \leq L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|, \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n \tag{8.2.7}$$

for some $L > 0$. The quantity L is known as the *Lipschitz constant* of ∇f. When this condition holds, a bit of calculus shows that we can complement the linear lower bound (8.2.6) with a corresponding quadratic upper bound:

Lemma 8.2.1. *Suppose that f is differentiable, and ∇f is L-Lipschitz. Then for every $\boldsymbol{x}, \boldsymbol{x}'$,*

$$f(\boldsymbol{x}') \leq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle + \frac{L}{2} \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 . \tag{8.2.8}$$

Proof. We calculate:

$$\begin{aligned}
f(\boldsymbol{x}') &= f(\boldsymbol{x} + t(\boldsymbol{x}' - \boldsymbol{x}))|_{t=1} && (8.2.9)\\
&= f(\boldsymbol{x}) + \int_{t=0}^{1} \frac{d}{dt} f(\boldsymbol{x} + t(\boldsymbol{x}' - \boldsymbol{x}))\, dt && (8.2.10)\\
&= f(\boldsymbol{x}) + \int_{t=0}^{1} \langle \nabla f(\boldsymbol{x} + t(\boldsymbol{x}' - \boldsymbol{x})), \boldsymbol{x}' - \boldsymbol{x} \rangle \, dt && (8.2.11)\\
&= f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle \\
&\quad + \int_{t=0}^{1} \langle \nabla f(\boldsymbol{x} + t(\boldsymbol{x}' - \boldsymbol{x})) - \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle \, dt && (8.2.12)\\
&\leq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle + \int_{t=0}^{1} tL \, \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 \, dt && (8.2.13)\\
&= f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle + \frac{L}{2} \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 , && (8.2.14)
\end{aligned}$$

giving the claim. □

Thus, when ∇f is Lipschitz, we have a matching quadratic upper bound

$$f(\boldsymbol{x}') \;\leq\; \hat{f}(\boldsymbol{x}', \boldsymbol{x}) \;\doteq\; f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle + \frac{L}{2} \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 , \tag{8.2.15}$$

and that the upper bound agrees with f at the point $\boldsymbol{x}$ at which it is formed: $f(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}, \boldsymbol{x})$. Suppose that we minimize this upper bound, with respect to $\boldsymbol{x}'$. A quick calculation shows that the minimizer has a very familiar form:

$$\arg\min_{\boldsymbol{x}'} \hat{f}(\boldsymbol{x}', \boldsymbol{x}) \;=\; \boldsymbol{x} - \frac{1}{L} \nabla f(\boldsymbol{x}). \tag{8.2.16}$$

This is simply a gradient descent step, taken from $\boldsymbol{x}$, with a special choice of step size $\gamma = 1/L$. Moreover, because $\hat{f}(\boldsymbol{x}, \boldsymbol{x}) = f(\boldsymbol{x})$, this minimization does not increase the objective function: if $\boldsymbol{x}'_\star \in \arg\min_{\boldsymbol{x}'} \hat{f}(\boldsymbol{x}', \boldsymbol{x})$, then

$$f(\boldsymbol{x}'_\star) \;\leq\; \hat{f}(\boldsymbol{x}'_\star, \boldsymbol{x}) \;\leq\; \hat{f}(\boldsymbol{x}, \boldsymbol{x}) \;=\; f(\boldsymbol{x}). \tag{8.2.17}$$

Thus, if we apply the gradient descent method with step size $1/L$, we are guaranteed to produce a monotone sequence of function values $f(\boldsymbol{x}_k)$. Furthermore, we can show convergence[3] to the optimal function value at a rate of $O(1/k)$:

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star) \;\leq\; \frac{L\,\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2k} \;=\; O(1/k). \tag{8.2.18}$$

This is still not a particularly fast rate of convergence, but it is much better than the $O(1/\sqrt{k})$ rate of convergence experienced by the subgradient algorithm on nonsmooth functions.

From gradient to proximal gradient.

Can we draw inspiration from the gradient method to produce a more efficient algorithm for minimizing the composite function $F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$, with f differentiable? Again, the gradient method does not directly apply, since F is nondifferentiable. Nevertheless, if the gradient ∇f of the smooth term is Lipschitz, we can still make a simpler upper bound to F, by upper bounding f by a quadratic and leaving the nonsmooth term g untouched:

$$\hat{F}(\boldsymbol{x}', \boldsymbol{x}) \;=\; f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}' - \boldsymbol{x} \rangle + \frac{L}{2}\,\|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 + g(\boldsymbol{x}'). \tag{8.2.19}$$

Since above repeatedly minimizing $\hat{f}$ produced the gradient method, resulting in a better convergence rate, let us try minimizing $\hat{F}$:

$$\boldsymbol{x}_{k+1} \;=\; \arg\min_{\boldsymbol{x}}\; \hat{F}(\boldsymbol{x}, \boldsymbol{x}_k). \tag{8.2.20}$$

For commonly encountered g, this minimization often takes on a very simple form. Completing the square in (8.2.19), we obtain that

$$\hat{F}(\boldsymbol{x}, \boldsymbol{x}_k) \;=\; \frac{L}{2}\left\|\boldsymbol{x} - (\boldsymbol{x}_k - \tfrac{1}{L}\nabla f(\boldsymbol{x}_k))\right\|_2^2 + g(\boldsymbol{x}) \;+\; h(\boldsymbol{x}_k), \tag{8.2.21}$$

where $h(\boldsymbol{x}_k)$ is a term that depends only on $\boldsymbol{x}_k$.

Hence, the iteration (8.2.20) becomes

$$\boldsymbol{x}_{k+1} \;=\; \arg\min_{\boldsymbol{x}}\; \frac{L}{2}\left\|\boldsymbol{x} - (\boldsymbol{x}_k - \tfrac{1}{L}\nabla f(\boldsymbol{x}_k))\right\|_2^2 + g(\boldsymbol{x}) \tag{8.2.22}$$

$$\;=\; \arg\min_{\boldsymbol{x}}\, g(\boldsymbol{x}) + \frac{L}{2}\,\|\boldsymbol{x} - \boldsymbol{w}_k\|_2^2\,, \tag{8.2.23}$$

where for convenience we define $\boldsymbol{w}_k = \boldsymbol{x}_k - (1/L)\nabla f(\boldsymbol{x}_k)$. Thus, at each step of the iteration (8.2.20), we have to minimize the function g plus a separable quadratic $\frac{L}{2}\,\|\boldsymbol{x} - \boldsymbol{w}\|_2^2$. In a sense, this problem asks us to make g as small as possible, while not straying too far from the point $\boldsymbol{w}$. Because

[3] We will not prove (8.2.18) here, although we will obtain a more general result below which implies it.

$\|\cdot\|_2^2$ is strongly convex, this problem always has a solution, and the solution is unique. So, the sequence $\boldsymbol{x}_k$ defined recursively by (8.2.20) is well-defined.

In fact, the operation of minimizing a convex function g plus a separable quadratic $\|\boldsymbol{x}-\boldsymbol{w}\|_2^2$ recurs so frequently in convex analysis and optimization that it has its own name. This is known as the *proximal operator* for the convex function $g(\boldsymbol{x})$:

Definition 8.2.2 (Proximal Operator)**.** *The proximal operator of a convex function g is*

$$\mathrm{prox}_g(\boldsymbol{w}) \doteq \arg\min_{\boldsymbol{x}} \left\{ g(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{w}\|_2^2 \right\}. \tag{8.2.24}$$

In this language, iteration (8.2.20) can be written as

$$\boldsymbol{x}_{k+1} = \mathrm{prox}_{g/L}(\boldsymbol{w}_k). \tag{8.2.25}$$

Fortunately, many of the convex functions (or norms) that we encounter in structured signal recovery either have close-form proximal operators or proximal operators that can be computed very efficiently via numerical means. We give a few examples below:

Proposition 8.2.3. *Proximal operators for the indicator function, ℓ^1 norm, and nuclear norm are given by:*

1. *Let $g(\boldsymbol{x}) = I_{\mathcal{D}}$ be the indicator function for a closed convex set $\mathcal{D}$, namely, $I_{\mathcal{D}}(\boldsymbol{x}) = 0, \boldsymbol{x} \in \mathcal{D}$ otherwise $I_{\mathcal{D}}(\boldsymbol{x}) = \infty$. Then $\mathrm{prox}_g(\boldsymbol{w})$ is the projection operator:*

$$\mathrm{prox}_g(\boldsymbol{w}) \;=\; \arg\min_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{x}-\boldsymbol{w}\|_2^2 \;=\; \mathcal{P}_{\mathcal{D}}[\boldsymbol{w}].$$

2. *Let $g(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$ be the ℓ^1 norm. Then $\mathrm{prox}_g(\boldsymbol{w})$ is the* soft-thresholding *function applied element-wise:*

$$\mathrm{prox}_g(u_i) = \mathrm{soft}(u_i, \lambda) \doteq \mathrm{sign}\,(u_i)\max(|u_i| - \lambda, 0).$$

3. *Let $g(\boldsymbol{x}) = \lambda\|\boldsymbol{X}\|_*$ be the matrix nuclear norm. Then $\mathrm{prox}_g(\boldsymbol{W})$ is the* singular-value thresholding *function:*

$$\mathrm{prox}_g(\boldsymbol{W}) = \mathcal{S}(\boldsymbol{W}, \lambda) \doteq \boldsymbol{U}\mathrm{soft}(\boldsymbol{\Sigma}, \lambda)\boldsymbol{V}^*,$$

where $(\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V})$ are the singular value decomposition (SVD) of $\boldsymbol{W}$. In other words, $\mathcal{S}(\boldsymbol{W}, \lambda)$ applies soft thresholding operator on the singular values of $\boldsymbol{X}$.

Proof. We prove the second assertion. The objective function reaches minimum when the subdifferential of $\lambda\|\boldsymbol{x}\|_1 + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{w}\|_2^2$ contains zero,

$$0 \in (\boldsymbol{x}-\boldsymbol{w}) + \lambda\partial\|\boldsymbol{x}\|_1 = \begin{cases} x_i - w_i + \lambda, & x_i > 0 \\ -w_i + \lambda[-1,1], & x_i = 0 \\ x_i - w_i - \lambda, & x_i < 0 \end{cases}, \quad i = 1, \ldots, n.$$

Therefore, the solution to this equality constraint is the soft-thresholding function applied element-wise:

$$x_{i\star} = \text{soft}(w_i, \lambda) \doteq \text{sign}\,(w_i)\max(|w_i| - \lambda, 0), \quad i = 1, \ldots, n.$$

We leave the first and third assertions as exercises to the reader. [*Hint:* for the first, use the definition; for the third, use the subdifferential of $\|\cdot\|_*$.] □

Thus, for problems of our interest, we can compute the proximal operator efficiently. In this setting, it provides an alternative replacement for the gradient step. Unlike the subgradient method, this *proximal gradient* algorithm enjoys a convergence rate of $O(1/k)$ – exactly the same as if the nonsmooth term was not present! More formally:

Theorem 8.2.4. *Let $F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$, with f be a convex, differentiable function with L-Lipschitz continuous gradient, and g a convex function. Consider the following iterative update scheme:*

$$\boldsymbol{w}_k \leftarrow \boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k), \quad \boldsymbol{x}_{k+1} \leftarrow \text{prox}_{g/L}\,(\boldsymbol{w}_k)\,.$$

Assume $F(\boldsymbol{x})$ has a minimum at $\boldsymbol{x}_\star$. Then for any $k \geq 1$,

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star) \leq \frac{L\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2k}.$$

We will give a detailed proof to this theorem in Section 8.2.3. Thus, for a composite non-smooth convex function, under certain conditions, we can still obtain an efficient "gradient descent" like algorithm that has the same convergence rate $O(1/k)$ as that for a smooth function. As long as the non-smooth part has an easy-to-solve proximal operator, the proximal gradient algorithm has very cheap iterations hence it is typically much more scalable than second-order methods. We summarize properties of the iterative process that we have derived so far for minimizing the convex composite problem in Figure 8.1, which is also known as the *proximal gradient* method.

8.2.1 Proximal Gradient for the Lasso

For the rest of the section, we will see how to apply the proximal gradient algorithm to several important cases of structured signal recovery problems that we have encountered.

As the first instance, the Lasso problem (8.1.2) obviously falls into the class of problems that can be addressed by the proximal gradient method. We can view g to be the ℓ^1 norm function $\lambda\|\boldsymbol{x}\|_1$ whose proximal operator is given in Proposition 8.2.3; f is simply be the quadratic data term $\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ whose gradient is clearly Lipschitz: the Lipschitz constant L can be the largest eigenvalue of the matrix $\boldsymbol{A}^*\boldsymbol{A}$, which can be computed in advance.

Proximal Gradient (PG)

Problem Class:

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$$

$f, g : \mathbb{R}^n \to \mathbb{R}$ are convex, ∇f L-Lipschitz and g non-smooth.

Basic Iteration: set $\boldsymbol{x}_0 \in \mathbb{R}^n$.
Repeat:

$$\boldsymbol{w}_k \leftarrow \boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k),$$
$$\boldsymbol{x}_{k+1} \leftarrow \operatorname{prox}_{g/L}(\boldsymbol{w}_k).$$

Convergence Guarantee:

$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star)$ converges at a rate of $O(1/k)$.

Figure 8.1. Properties of the Proximal Gradient Method.

The resulting proximal gradient descent algorithm for Lasso is sometimes referred to as the *iterative soft-thresholding algorithm* (ISTA), which is summarized in Algorithm 8.1. In terms of computational complexity, the

Algorithm 8.1 Proximal Gradient (PG) for Lasso

1: **Problem:** $\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$, given $\boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$.
2: **Input:** $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $L \geq \lambda_{\max}(\boldsymbol{A}^*\boldsymbol{A})$.
3: **while** $\boldsymbol{x}_k$ not converged $(k = 1, 2, \ldots)$ **do**
4: $\quad \boldsymbol{w}_k \leftarrow \boldsymbol{x}_k - \frac{1}{L}\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})$.
5: $\quad \boldsymbol{x}_{k+1} \leftarrow \operatorname{soft}(\boldsymbol{w}_k, \lambda/L)$.
6: **end while**
7: **Output:** $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

main cost is calculating the gradient $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^*\boldsymbol{y}$ in the inner loop, which is in the order of $O(mn)$ in general.

Example 8.2.5. *We randomly generate a sparse signal $\boldsymbol{x}$ in an 1000-D space and then add a small Gaussian noise $\boldsymbol{n}$ to all of its coefficients, as shown in Figure 8.2 Top. With the added Gaussian noise, the signal $\boldsymbol{w} = \boldsymbol{x} + \boldsymbol{n}$ is not sparse anymore. Then we may try to recover $\boldsymbol{x}$ from $\boldsymbol{w}$ by solving the following problem $\min_{\boldsymbol{x}} \lambda\|\boldsymbol{x}\|_1 + \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{x}\|_2^2$, where λ is proportional to the noise level. We know the solution to this problem is simply the soft thresholding $\hat{\boldsymbol{x}} = \operatorname{soft}(\boldsymbol{w}, \lambda)$. The result is shown in Figure*

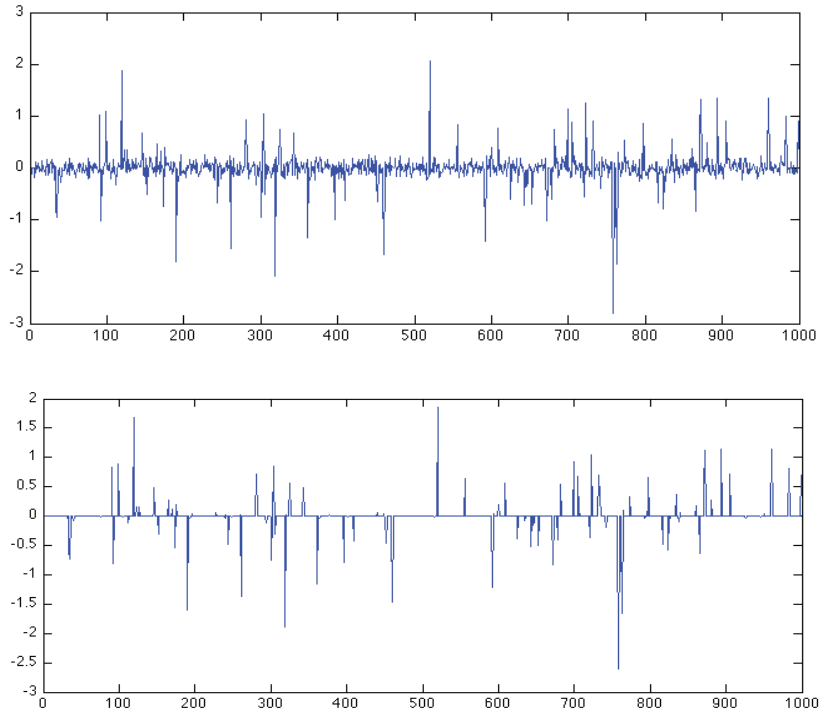

Figure 8.2. **Top:** A sparse signal $\boldsymbol{x}$ perturbed by small Gaussian noise $\boldsymbol{n}$. **Bottom:** The output from a properly chosen soft-thresholding function $\text{soft}(\boldsymbol{w}, \lambda)$.

8.2 Bottom. We see that the operator successfully remove most of the noise in $\boldsymbol{w}$ and returns a sparse estimate for $\boldsymbol{x}$.

8.2.2 Proximal Gradient for Stable PCP

According to Proposition 8.2.3, the nuclear norm $\|\boldsymbol{X}\|_*$ also has a simple proximal operator. Hence we could apply proximal gradient algorithm to solve low-rank matrix recovery problems. For instance, the stable principal component pursuit (PCP) program is also for the form that is amenable to proximal gradient method:

$$\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \frac{\mu}{2}\|\boldsymbol{Y} - \boldsymbol{L} - \boldsymbol{S}\|_F^2. \tag{8.2.26}$$

Notice that however, for this problem the non-smooth term $g(\boldsymbol{L}, \boldsymbol{S}) = \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1$ now contains two non-smooth functions $\|\boldsymbol{L}\|_*$ and $\lambda\|\boldsymbol{S}\|_1$, each having a simple proximal operator.

We leave as an exercise for the reader to prove the following simple fact about the proximal operator of a separable convex function, which comes handy for this problem: Let $\boldsymbol{x} = [\boldsymbol{x}_1; \boldsymbol{x}_2]$ and $g(\boldsymbol{x}) = g_1(\boldsymbol{x}_1) + g_2(\boldsymbol{x}_2)$ be a separable function. Then

$$\operatorname{prox}_g(\boldsymbol{w}) = \left(\operatorname{prox}_{g_1}(\boldsymbol{w}_1), \operatorname{prox}_{g_2}(\boldsymbol{w}_2)\right),$$

where $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ in $\boldsymbol{w} = [\boldsymbol{w}_1; \boldsymbol{w}_2]$ correspond to the variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in $\boldsymbol{x}$, respectively.

Hence, the proximal operator for $g(\boldsymbol{L}, \boldsymbol{S})$ can be computed separately from the proximal operators for the ℓ^1 norm for for $\boldsymbol{S}$ and nuclear norm for $\boldsymbol{L}$, respectively. The rest of the proximal gradient algorithm for the stable principal component pursuit program then is easy to derive (and we leave this as an exercise to the reader. See Exercise 8.2.). We summarize the overall algorithm below for clarity.

Algorithm 8.2 Proximal Gradient (PG) for Stable Principal Component Pursuit

Problem: $\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \frac{\mu}{2}\|\boldsymbol{Y} - \boldsymbol{L} - \boldsymbol{S}\|_F^2$, given $\boldsymbol{Y}$, $\lambda, \mu > 0$.

Input: $\boldsymbol{L}_0 \in \mathbb{R}^{m\times n}$, $\boldsymbol{S}_0 \in \mathbb{R}^{m\times n}$.
while $((\boldsymbol{L}_k, \boldsymbol{S}_k)$ not converged $(k = 1, 2, \ldots)$ **do**
  $\boldsymbol{W}_k \leftarrow \boldsymbol{Y} - \boldsymbol{S}_k$ and compute $\boldsymbol{W}_k = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$.
  $\boldsymbol{L}_{k+1} \leftarrow \boldsymbol{U}\operatorname{soft}(\boldsymbol{\Sigma}, 1/\mu)\boldsymbol{V}^*$.
  $\boldsymbol{S}_{k+1} \leftarrow \operatorname{soft}((\boldsymbol{Y} - \boldsymbol{L}_k), \lambda/\mu)$.
end while
Output: $\boldsymbol{L}_\star \leftarrow \boldsymbol{L}_k; \boldsymbol{S}_\star \leftarrow \boldsymbol{S}_k$.

8.2.3 Convergence of Proximal Gradient

In this subsection, we prove Theorem 8.2.4. We find it convenient to do this in two steps. In the first step, we proved an analysis of a simpler algorithm, known as the *proximal point algorithm*, which only consists of repeated application of the proximal operator. This algorithm is of independent interest, and we will reuse its analysis when we encounter Augmented Lagrangian techniques.

Proposition 8.2.6 (Proximal point algorithm)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a convex function, and $\boldsymbol{x}_\star$ a minimizer of g. Let $\boldsymbol{x}_0 \in \mathbb{R}^n$ be arbitrary, and consider the iteration*

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_{\gamma_k g}(\boldsymbol{x}_k), \tag{8.2.27}$$

with $\gamma_k \in \mathbb{R}_+$. *Then*

$$g(\boldsymbol{x}_k) - g(\boldsymbol{x}_\star) \;\leq\; \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2\sum_{i=1}^k \gamma_i}. \tag{8.2.28}$$

Moreover, if $\sum_{i=1}^\infty \gamma_i = +\infty$, *then* $\boldsymbol{x}_k \to \bar{\boldsymbol{x}}$, *a minimizer of* g.

Proof. By construction,

$$\mathbf{0} \in \gamma_k \partial g(\boldsymbol{x}_{k+1}) + \boldsymbol{x}_{k+1} - \boldsymbol{x}_k. \tag{8.2.29}$$

Equivalently, $\boldsymbol{x}_k - \boldsymbol{x}_{k+1} \in \gamma_k \partial g(\boldsymbol{x}_{k+1})$. Using the subgradient inequality, we have that

$$\langle \boldsymbol{x}_k - \boldsymbol{x}_{k+1}, \boldsymbol{x}_k - \boldsymbol{x}_{k+1} \rangle \leq \gamma_k \left(g(\boldsymbol{x}_k) - g(\boldsymbol{x}_{k+1}) \right). \tag{8.2.30}$$

Since left hand side is nonnegative, $g(\boldsymbol{x}_k) \geq g(\boldsymbol{x}_{k+1})$. So, the objective function is nondecreasing.

Using the subgradient inequality again, we have that

$$\langle \boldsymbol{x}_\star - \boldsymbol{x}_{k+1}, \boldsymbol{x}_k - \boldsymbol{x}_{k+1} \rangle \leq \gamma_k \left(g(\boldsymbol{x}_\star) - g(\boldsymbol{x}_{k+1}) \right). \tag{8.2.31}$$

Let us use this fact to bound the distance of $\boldsymbol{x}_{k+1}$ to the optimum. Notice that

$$\begin{aligned}
\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star\|_2^2 &= \|\boldsymbol{x}_k - \boldsymbol{x}_\star + \boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2 \\
&= \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2^2 + 2\langle \boldsymbol{x}_k - \boldsymbol{x}_\star, \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle + \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2 \\
&= \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2 + 2\langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_\star, \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle \\
&\leq \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2^2 + 2\langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_\star, \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \rangle \\
&\leq \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2^2 + 2\gamma_k \left(g(\boldsymbol{x}_\star) - g(\boldsymbol{x}_{k+1}) \right).
\end{aligned} \tag{8.2.32}$$

Since $g(\boldsymbol{x}_{k+1}) \geq g(\boldsymbol{x}_\star)$, the distance of $\boldsymbol{x}_k$ to $\boldsymbol{x}_\star$ also does not increase. In fact, we can say slightly more. Summing the relationship (8.2.32), we obtain

$$\sum_{i=1}^k 2\gamma_i \left(g(\boldsymbol{x}_{i+1}) - g(\boldsymbol{x}_\star) \right) \;\leq\; \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2. \tag{8.2.33}$$

Since $g(\boldsymbol{x}_i)$ is nonincreasing, this implies that

$$2\left(\sum_{i=1}^k \gamma_i \right) \left(g(\boldsymbol{x}_{k+1}) - g(\boldsymbol{x}_\star) \right) \;\leq\; \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2. \tag{8.2.34}$$

This gives convergence in function values, as in (8.2.28).

Since $\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2$ is nonincreasing, the sequence $\boldsymbol{x}_k$ is bounded, and hence has a cluster point $\bar{\boldsymbol{x}}$. Since $g(\boldsymbol{x}_k) \searrow g(\boldsymbol{x}_\star)$, $g(\bar{\boldsymbol{x}}) = g(\boldsymbol{x}_\star)$, and hence $\bar{\boldsymbol{x}}$ is optimal. Applying inequality (8.2.32) with $\boldsymbol{x}_\star$ replaced by $\bar{\boldsymbol{x}}$, we obtain that $\|\boldsymbol{x}_k - \bar{\boldsymbol{x}}\|_2$ is also nonincreasing, whence the cluster point $\bar{\boldsymbol{x}}$ is the limit of the sequence $\{\boldsymbol{x}_k\}$. □

The key idea in the above proof is to use the optimality condition (8.2.29) for the proximal operator, together with the subgradient inequality to relate the suboptimality $g(\boldsymbol{x}_{k+1}) - g(\boldsymbol{x}_\star)$ in objective function to the distance to the feasible set. To prove Theorem 8.2.4, we follow a very similar program.

Proof. Notice that by construction, there exists $\boldsymbol{\gamma} \in \partial g(\boldsymbol{x}_{k+1})$ such that

$$\nabla f(\boldsymbol{x}_k) + L(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \boldsymbol{\gamma} = \mathbf{0}. \tag{8.2.35}$$

The subgradient and gradient inequalities for the convex functions f and g give that for any $\boldsymbol{x}$,

$$\begin{aligned} f(\boldsymbol{x}) &\geq f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle , & (8.2.36) \\ g(\boldsymbol{x}) &\geq g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \boldsymbol{\gamma} \rangle , & (8.2.37) \end{aligned}$$

whence

$$F(\boldsymbol{x}) \geq f(\boldsymbol{x}_k) + g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle + \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \boldsymbol{\gamma} \rangle . \tag{8.2.38}$$

So,

$$\begin{aligned} F(\boldsymbol{x}) - F(\boldsymbol{x}_{k+1}) &\geq F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k) \\ &\geq f(\boldsymbol{x}_k) + g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle \\ &\quad + \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \boldsymbol{\gamma} \rangle - \hat{F}(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k) \\ &= -\frac{L}{2} \left\| \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \right\|_2^2 + \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \nabla f(\boldsymbol{x}_k) + \boldsymbol{\gamma} \rangle \\ &= -\frac{L}{2} \left\| \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \right\|_2^2 + L \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \boldsymbol{x}_k - \boldsymbol{x}_{k+1} \rangle \\ &= \frac{L}{2} \left\| \boldsymbol{x} - \boldsymbol{x}_{k+1} \right\|_2^2 - \frac{L}{2} \left\| \boldsymbol{x} - \boldsymbol{x}_k \right\|_2^2 . \end{aligned} \tag{8.2.39}$$

Evaluating this expression at $\boldsymbol{x} = \boldsymbol{x}_\star$, we see that $\left\| \boldsymbol{x}_k - \boldsymbol{x}_\star \right\|_2$ is nonincreasing. Moreover, rearranging the relationship and summing from 0 to k, we obtain that

$$\frac{2}{L} \sum_{i=0}^{k-1} \{F(\boldsymbol{x}_{i+1}) - F(\boldsymbol{x}_\star)\} \leq \left\| \boldsymbol{x}_0 - \boldsymbol{x}_\star \right\|_2^2 . \tag{8.2.40}$$

Evaluating (8.2.39) at $\boldsymbol{x} = \boldsymbol{x}_k$, we obtain

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_{k+1}) \geq \frac{L}{2} \left\| \boldsymbol{x}_k - \boldsymbol{x}_{k+1} \right\|_2^2 . \tag{8.2.41}$$

Hence, (8.2.40) implies that

$$\frac{2}{L} k \{F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star)\} \leq \left\| \boldsymbol{x}_0 - \boldsymbol{x}_\star \right\|_2^2 . \tag{8.2.42}$$

Rearranging, we get the desired conclusion. □

8.3 Accelerated Proximal Gradient Methods

In the previous section, we have seen that by exploiting the fact that if a (non-smooth) convex function has an easily computable proximal operator, we are able to extend the gradient descent algorithm for smooth functions to the special class of composite objective functions that we encounter in structured signal recovery. The resulting algorithm enjoys the same $O(1/k)$ convergence rate as in the smooth case. Recognizing special structure in our problem of interest yields a significantly more accurate, efficient algorithm.

Interlude: Accelerating the gradient method.

With the taste of victory still on our lips, we might naturally ask whether there are further improvements to be had – is our proximal gradient algorithm optimal for this class of functions? For the question to be meaningful, we need to restrict our attention to methods with efficient iterations, such as gradient-like methods. For example, we could restrict our attention to *first order methods*, which base their future actions on the past iterates $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$, objective values at these iterates, and the gradients $\nabla f(\boldsymbol{x}_0), \ldots, \nabla f(\boldsymbol{x}_k)$. The corresponding question for *smooth functions* was studied in great depth in the late 1970's and early 1980's by Russian optimization theorists, including Nemirovski, Yudin, and Nesterov.[4]

They asked the very natural question: *for minimizing a smooth function f, is the gradient method optimal amongst first-order methods?* Again, to study this problem one needs a model for computation. They considered a *black box* model, in which the algorithm produces a sequence of iterates $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$. At each iteration, the algorithm is provided with the value $f(\boldsymbol{x}_i)$ and the gradient $\nabla f(\boldsymbol{x}_i)$. It produces the next iterate as a function of the history of iterates, gradients, and function values up to this time:

$$\boldsymbol{x}_{k+1} = \varphi_k \left(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k, f(\boldsymbol{x}_0), \ldots, f(\boldsymbol{x}_k), \nabla f(\boldsymbol{x}_0), \ldots, \nabla f(\boldsymbol{x}_k) \right). \quad (8.3.1)$$

With this model in mind, one can begin to study algorithms from a worst-case perspective. To do so, we fix a class of functions $\mathcal{F}$, and ask how well the algorithm does on the "worst function" from this class:

$$\sup_{f \in \mathcal{F},\, \boldsymbol{x}_0} \left\{ f(\boldsymbol{x}_k) - \inf_{\boldsymbol{x}} f(\boldsymbol{x}) \right\}. \quad (8.3.2)$$

One can study various classes of functions $\mathcal{F}$. However, for our purposes, one interesting class is the convex differentiable functions $f : \mathfrak{B}(0, r) \to \mathbb{R}$, with L-Lipschitz gradient:

$$\mathcal{F}_{L,r} \doteq \{ f : \mathfrak{B}(0,r) \to \mathbb{R} \mid \|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x}')\| \leq L \|\boldsymbol{x} - \boldsymbol{x}'\| \; \forall \, \boldsymbol{x}, \boldsymbol{x}' \in \mathfrak{B}(0,r) \}. \quad (8.3.3)$$

[4] For a more comprehensive introduction to this circle of ideas, see [Nesterov, 2003a] or [Nemirovski, 1995].

The gradient method achieves a rate of $O(1/k)$ over this class:

$$\sup_{f \in \mathcal{F}_{L,r},\, \boldsymbol{x}_0} \Big\{ f(\boldsymbol{x}_k) - \inf_{\boldsymbol{x}} f(\boldsymbol{x}) \Big\} \;\leq\; \frac{CLr^2}{k}. \tag{8.3.4}$$

However, the best lower bound that anyone could prove was of much lower order:

$$\sup_{f \in \mathcal{F}_{L,r},\, \boldsymbol{x}_0} \Big\{ f(\boldsymbol{x}_k) - \inf_{\boldsymbol{x}} f(\boldsymbol{x}) \Big\} \;\geq\; \frac{cLr^2}{k^2}. \tag{8.3.5}$$

Was this merely a gap in the theory? Or might there actually exist a "faster" gradient method than gradient descent itself?

In 1983, Yuri Nesterov closed this gap, to remarkable effect [Nesterov, 1983]. He demonstrated a relatively simple first-order method, which achieved the optimal rate of convergence, $O(1/k^2)$. The analysis of this algorithm is straightforward to read – the 1983 paper is only five pages! However, it is not straightforward to build intuition into *how* the method achieves this rate. To gain some loose appreciation for what is going on, we start from a simpler idea.

Let us first consider the gradient descent method for a smooth function with Lipschitz gradient. At each iteration, the update simply follows the direction of the gradient:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k),$$

where a good choice of α for an L-Lipschitz function is $1/L$. The gradient $-\nabla f(\boldsymbol{x}_k)$ indicates the direction in which the function drops it value the fastest. Instead of updating along this most greedy direction at the current estimate $\boldsymbol{x}_k$, a slightly more conservative strategy is to update by keeping some momentum from the previous update direction: $\boldsymbol{x}_k - \boldsymbol{x}_{k+1}$. That leads an update rule that is known as the *heavy ball method*:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) + \beta\big(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\big). \tag{8.3.6}$$

The heavy ball method often leads to smoother update trajectories and faster convergence.

Like the heavy ball method, Nesterov's method uses a momentum step. It introduces an auxiliary point $\boldsymbol{p}_{k+1}$ of the form similar to that in the heavy ball method:

$$\boldsymbol{p}_{k+1} \doteq \boldsymbol{x}_k + \beta_k\big(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\big).$$

At each iteration, we move to this new point, and then descend from it:

$$\boldsymbol{x}_{k+1} = \boldsymbol{p}_{k+1} - \alpha \nabla f(\boldsymbol{p}_{k+1}). \tag{8.3.7}$$

The weights $\alpha = 1/L$ and β_k are carefully chosen to achieve the optimal convergence rate:

$$t_k = \frac{1 + \sqrt{1 + 4t_{k-1}^2}}{2}, \quad \beta_k = \frac{t_{k-1} - 1}{t_k}. \tag{8.3.8}$$

These particular values come from the convergence analysis. One can rigorously show that with this update scheme, the resulting algorithm achieves the theoretically optimal convergence rate $O(1/k^2)$ for the class of smooth functions with Lipschitz gradient.

Accelerating the proximal *gradient method.*

As we have seen in the previous section, convex programs that arise in the context of structured signal recovery are often of the composite form $F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$, where f is a smooth term whose gradient is L-Lipschitz and $g(\boldsymbol{x})$ is convex but not necessarily smooth. In the proximal gradient method introduced in the previous section, we have shown that at the k-th iteration, the value of the objective function $F(\boldsymbol{x})$ can be upper-bounded by

$$\begin{aligned} \hat{F}(\boldsymbol{x}, \boldsymbol{x}_k) &\doteq f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{x} - \boldsymbol{x}_k \rangle + \tfrac{L}{2}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 + g(\boldsymbol{x}) \\ &\doteq \tfrac{L}{2}\|\boldsymbol{x} - \boldsymbol{w}_k\|_2^2 + g(\boldsymbol{x}) \ + \ \text{terms that do not depend on } \boldsymbol{x}, \end{aligned}$$

where $\boldsymbol{w}_k = \boldsymbol{x}_k - \frac{1}{L}\nabla f(\boldsymbol{x}_k)$. As we have seen in the previous section, the gradient descent algorithm for the smooth part $f(\boldsymbol{x})$ corresponds to directly minimizing its quadratic approximation of $f(\boldsymbol{x})$ at $\boldsymbol{x}_k$.

In this langauge, if $g \equiv 0$, Nesterov's method corresponds to: *extrapolating* to find the point $\boldsymbol{p}_{k+1}$ based on the past two iterates, and then *minimizing* a quadratic upper bound $\hat{f}$ to f, taken at the new point $\boldsymbol{p}_{k+1}$, by taking a gradient step. Let us attempt to do the same thing for our composite function $\hat{F}$. Set

$$\boldsymbol{p}_{k+1} = \boldsymbol{x}_k + \beta_k\big(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\big), \tag{8.3.9}$$

instead of the current estimate $\boldsymbol{x}_k$. Then minimize $\hat{F}(\boldsymbol{x}, \boldsymbol{p}_{k+1})$ to obtain the next iterate $\boldsymbol{x}_{k+1}$:

$$\boldsymbol{x}_{k+1} = \operatorname{prox}_g\Big(\boldsymbol{p}_{k+1} - \frac{1}{L}\nabla f(\boldsymbol{p}_{k+1})\Big). \tag{8.3.10}$$

We summarize this scheme in Figure 8.3. Theorem 8.3.1 establishes that with this simple modification to the proximal gradient method, the resulting new algorithm achieves the theoretically optimal convergence rate $O(1/k^2)$ for this class of methods, despite the presence of a nonsmooth term in the objective function.

Theorem 8.3.1. *Let the sequence $\{\boldsymbol{x}_k\}$ be generated by the above accelerated proximal gradient scheme for the convex composite function $F(\boldsymbol{x}) =$*

Accelerated Proximal Gradient (APG)

Problem Class:

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$$

$f, g : \mathbb{R}^n \to \mathbb{R}$ are convex, ∇f L-Lipschitz and g non-smooth.

Basic Iteration: set $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\boldsymbol{p}_1 \leftarrow \boldsymbol{x}_0$, and $t_0 \leftarrow 1$.
Repeat:

$$t_k \leftarrow \frac{1 + \sqrt{1 + 4{t_{k-1}}^2}}{2}, \quad \beta_k \leftarrow \frac{t_{k-1} - 1}{t_k}.$$
$$\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{x}_k + \beta_k\big(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\big).$$
$$\boldsymbol{x}_{k+1} \leftarrow \mathrm{prox}_g\Big(\boldsymbol{p}_{k+1} - \frac{1}{L}\nabla f(\boldsymbol{p}_{k+1})\Big).$$

Convergence Guarantee:

$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star)$ converges at a rate of $O(1/k^2)$.

Figure 8.3. Properties of the Accelerated Proximal Gradient Method.

$f(\boldsymbol{x}) + g(\boldsymbol{x})$, where the gradient of f is L-Lipschitz. Let $\boldsymbol{x}_\star$ be a minimizer of $F(\boldsymbol{x})$. Then for any $k \geq 1$,

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star) \leq \frac{2L\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{(k+1)^2}.$$

8.3.1 APG for Basis Pursuit Denoising

Applying the APG algorithm in Figure 8.3 to the basis pursuit denoising problem 8.1.2, we obtain Algorithm 8.3.

Algorithm 8.3 Accelerated Proximal Gradient (APG) for BPDN

Problem: $\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$, given $\boldsymbol{y} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$.
Input: $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\boldsymbol{p}_1 \leftarrow \boldsymbol{x}_0$, and $t_1 \leftarrow 1$, and $L \geq \lambda_{\max}(\boldsymbol{A}^*\boldsymbol{A})$.
while $\boldsymbol{x}_k$ not converged $(k = 1, 2, \ldots)$ **do**
  $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{p}_k - \frac{1}{L}\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{p}_k - \boldsymbol{y})$.
  $\boldsymbol{x}_{k+1} \leftarrow \mathrm{soft}(\boldsymbol{w}_{k+1}, \lambda/L)$.
  $t_{k+1} \leftarrow \frac{1+\sqrt{1+4t_k^2}}{2}$; $\beta_{k+1} \leftarrow \frac{t_k - 1}{t_{k+1}}$.
  $\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{x}_{k+1} + \beta_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$.
end while
Output: $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

8.3.2 APG for Stable Principal Component Pursuit

Similarly we could apply the APG algorithm in Figure 8.3 to solving the stable principal component pursuit (PCP) problem 8.2.26. Again, notice that the APG scheme respects the natural separable structure of the objective function.

Algorithm 8.4 Accelerated Proximal Gradient (APG) for Stable PCP

Problem: $\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \frac{\mu}{2}\|\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}\|_F^2$, given $\boldsymbol{Y}$, $\lambda, \mu > 0$.
Input: $\boldsymbol{L}_0 \in \mathbb{R}^{m\times n}$, $\boldsymbol{S}_0 \in \mathbb{R}^{m\times n}$, $t_1 \leftarrow 1$.
while $(\boldsymbol{L}_k, \boldsymbol{S}_k)$ not converged $(k = 1, 2, \ldots)$ **do**
  $t_k \leftarrow \frac{1+\sqrt{1+4t_{k-1}^2}}{2}$, $\beta_k \leftarrow \frac{t_{k-1}-1}{t_k}$.
  $\boldsymbol{P}_k^L \leftarrow \boldsymbol{L}_k + \beta_k(\boldsymbol{L}_k - \boldsymbol{L}_{k-1})$.
  $\boldsymbol{P}_k^S \leftarrow \boldsymbol{S}_k + \beta_k(\boldsymbol{S}_k - \boldsymbol{S}_{k-1})$.
  $\boldsymbol{W}_k \leftarrow \boldsymbol{Y} - \boldsymbol{P}_k^S$ and compute the SVD: $\boldsymbol{W}_k = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$.
  $\boldsymbol{L}_{k+1} \leftarrow \boldsymbol{U}\mathrm{soft}(\boldsymbol{\Sigma}, 1/\mu)\boldsymbol{V}^*$.
  $\boldsymbol{S}_{k+1} \leftarrow \mathrm{soft}((\boldsymbol{Y} - \boldsymbol{P}_k^L), \lambda/\mu)$.
end while
Output: $\boldsymbol{L}_\star \leftarrow \boldsymbol{L}_k; \boldsymbol{S}_\star \leftarrow \boldsymbol{S}_k$.

8.3.3 Convergence of APG

Our convergence analysis follows, almost verbatim, Beck and Teboulle [Beck and Teboulle, 2009]. Let $\varphi(\boldsymbol{y})$ denote the operator that takes a step in the direction of the gradient of f at $\boldsymbol{y}$, and then applies the proximal operator of g/L:

$$\varphi(\boldsymbol{y}) = \mathrm{prox}_{g/L}\left(\boldsymbol{y} - \frac{1}{L}\nabla f(\boldsymbol{y})\right). \tag{8.3.11}$$

In this language, the accelerated proximal gradient iteration is

$$\boldsymbol{x}_{k+1} = \varphi(\boldsymbol{p}_{k+1}). \tag{8.3.12}$$

The following lemma allows us to compare the value of $F(\boldsymbol{x})$ at any point $\boldsymbol{x}$ to the value at $\varphi(\boldsymbol{p})$ for an arbitrary point $\boldsymbol{p}$:

Lemma 8.3.2. *For every $\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n$,*

$$F(\boldsymbol{x}) - F(\varphi(\boldsymbol{p})) \;\geq\; \frac{L}{2}\|\varphi(\boldsymbol{p}) - \boldsymbol{p}\|_2^2 + L\,\langle \boldsymbol{p} - \boldsymbol{x}, \varphi(\boldsymbol{p}) - \boldsymbol{p}\rangle\,. \tag{8.3.13}$$

Proof. For it, we note that from the optimality condition for the proximal problem, $\boldsymbol{z} = \varphi(\boldsymbol{p})$ if and only if there exists $\boldsymbol{\gamma} \in \partial g(\boldsymbol{z})$ such that

$$\boldsymbol{\gamma} + L(\boldsymbol{z} - \boldsymbol{p}) + \nabla f(\boldsymbol{p}) = \boldsymbol{0}. \tag{8.3.14}$$

Using the subgradient inequalities for f and g, we obtain that

$$\begin{aligned} f(\boldsymbol{x}) &\geq f(\boldsymbol{p}) + \langle \boldsymbol{x} - \boldsymbol{p}, \nabla f(\boldsymbol{p}) \rangle && (8.3.15) \\ g(\boldsymbol{x}) &\geq g(\varphi(\boldsymbol{p})) + \langle \boldsymbol{x} - \varphi(\boldsymbol{p}), \boldsymbol{\gamma} \rangle \,. && (8.3.16) \end{aligned}$$

Hence,

$$\begin{aligned} F(\boldsymbol{x}) - F(\varphi(\boldsymbol{p})) &\geq f(\boldsymbol{x}) + g(\varphi(\boldsymbol{p}) + \langle \boldsymbol{x} - \boldsymbol{p}, \nabla f(\boldsymbol{p}) \rangle \\ &\quad + \langle \boldsymbol{x} - \varphi(\boldsymbol{p}), \boldsymbol{\gamma} \rangle - F(\varphi(\boldsymbol{p})) \\ &\geq f(\boldsymbol{x}) + g(\varphi(\boldsymbol{p}) + \langle \boldsymbol{x} - \boldsymbol{p}, \nabla f(\boldsymbol{p}) \rangle \\ &\quad + \langle \boldsymbol{x} - \varphi(\boldsymbol{p}), \boldsymbol{\gamma} \rangle - \hat{F}(\boldsymbol{\varphi}(\boldsymbol{p}), p) \\ &= -\frac{L}{2} \left\| \varphi(\boldsymbol{p}) - \boldsymbol{p} \right\|_2^2 + \langle \boldsymbol{x} - \varphi(\boldsymbol{p}), \nabla f(\boldsymbol{p}) + \boldsymbol{\gamma} \rangle \\ &= -\frac{L}{2} \left\| \varphi(\boldsymbol{p}) - \boldsymbol{p} \right\|_2^2 + L \left\langle \boldsymbol{x} - \varphi(\boldsymbol{p}), \boldsymbol{p} - \varphi(\boldsymbol{p}) \right\rangle \\ &= \frac{L}{2} \left\| \varphi(\boldsymbol{p}) - \boldsymbol{p} \right\|_2^2 + L \left\langle \boldsymbol{p} - \boldsymbol{x}, \varphi(\boldsymbol{p}) - \boldsymbol{p} \right\rangle , \quad (8.3.17) \end{aligned}$$

as desired. □

Using Lemma 8.3.2, we obtain a relationship betwen the suboptimalities in function values and the distance of an interpolated point to the optimum:

Lemma 8.3.3. *Let $(\boldsymbol{x}_k, \boldsymbol{p}_k)$ be the sequence generated by the proximal gradient method. Set*

$$v_k = F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star) \tag{8.3.18}$$

and

$$\boldsymbol{u}_k = t_k \boldsymbol{x}_k - (t_k - 1)\boldsymbol{x}_{k-1} - \boldsymbol{x}_\star. \tag{8.3.19}$$

Then

$$\frac{2}{L} t_k^2 v_k - \frac{2}{L} t_{k+1}^2 v_{k+1} \;\geq\; \left\| \boldsymbol{u}_{k+1} \right\|_2^2 - \left\| \boldsymbol{u}_k \right\|_2^2 . \tag{8.3.20}$$

Proof. Let us apply the previous lemma with $\boldsymbol{x} = \boldsymbol{x}_k$ and $\boldsymbol{p} = \boldsymbol{p}_{k+1}$. This gives

$$\frac{2}{L} \left(v_k - v_{k+1} \right) \;\geq\; \left\| \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1} \right\|_2^2 + 2 \left\langle \boldsymbol{p}_{k+1} - \boldsymbol{x}_k, \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1} \right\rangle . \tag{8.3.21}$$

Applying the lemma with $\boldsymbol{x} = \boldsymbol{x}_\star$ and $\boldsymbol{p} = \boldsymbol{p}_{k+1}$, we get

$$-\frac{2}{L} v_{k+1} \;\geq\; \left\| \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1} \right\|_2^2 + 2 \left\langle \boldsymbol{p}_{k+1} - \boldsymbol{x}_\star, \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1} \right\rangle . \tag{8.3.22}$$

Combining these two inequalities, we get

$$\begin{aligned} &\frac{2}{L} \left((t_{k+1} - 1) v_k - t_{k+1} v_{k+1} \right) \\ \geq\; & t_{k+1} \left\| \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1} \right\|_2^2 + 2 \left\langle \boldsymbol{x}_{k+1} - \boldsymbol{p}_{k+1}, t_{k+1} \boldsymbol{p}_{k+1} - (t_{k+1} - 1) \boldsymbol{x}_k - \boldsymbol{x}_\star \right\rangle . \end{aligned}$$

Multiplying both sides by t_{k+1}, and using that $t_k^2 = t_{k+1}(t_{k+1}-1)$, we get

$$\begin{aligned}
&\frac{2}{L}\left(t_k^2 v_k - t_{k+1}^2 v_{k+1}\right) \\
\geq\ & \left\|t_{k+1}(\boldsymbol{x}_{k+1}-\boldsymbol{p}_{k+1})\right\|_2^2 + 2t_{k+1}\left\langle \boldsymbol{x}_{k+1}-\boldsymbol{p}_{k+1}, t_{k+1}\boldsymbol{p}_{k+1}-(t_{k+1}-1)\boldsymbol{x}_k-\boldsymbol{x}_\star\right\rangle \\
=\ & \left\|t_{k+1}\boldsymbol{x}_{k+1}-(t_{k+1}-1)\boldsymbol{x}_k-\boldsymbol{x}_\star\right\|_2^2 - \left\|\boldsymbol{t}_{k+1}\boldsymbol{p}_{k+1}-(t_{k+1}-1)\boldsymbol{x}_k-\boldsymbol{x}_\star\right\|_2^2 \\
=\ & \|\boldsymbol{u}_{k+1}\|_2^2 - \|\boldsymbol{u}_k\|_2^2,
\end{aligned}$$

as desired. □

To prove the desired result, we note two simple facts, and then perform a calculation. First,

Lemma 8.3.4. *Let (a_k, b_k) be sequences of positive real numbers satsifying*

$$a_k - a_{k+1} \ \geq\ b_{k+1} - b_k \ \forall\, k, \qquad a_1 + b_1 \leq c. \tag{8.3.23}$$

Then $a_k \geq c$ for every k.

Second,

Lemma 8.3.5. *The sequence t_k generated by the accelerated proximal gradient method satisfies*

$$t_k \ \geq\ \frac{k+1}{2} \quad \forall\, k \geq 1. \tag{8.3.24}$$

With these facts in mind, we calculate:

Proof. Define

$$a_k \ =\ \frac{2}{L}t_k^2 v_k \tag{8.3.25}$$

$$b_k \ =\ \|\boldsymbol{u}_k\|_2^2 \tag{8.3.26}$$

$$c \ =\ \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2\,. \tag{8.3.27}$$

By Lemma 8.3.3, for every k,

$$a_k - a_{k+1} \geq b_{k+1} - b_k \tag{8.3.28}$$

so, provided $a_1 + b_1 \geq c$, we obtain that $a_k \leq c$ for every k, whence

$$\frac{2}{L}t_k^2 v_k \ \leq\ \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2\,. \tag{8.3.29}$$

Since $t_k \geq (k+1)/2$, this gives

$$F(\boldsymbol{x}_k) - F(\boldsymbol{x}_\star) \ \leq\ \frac{2L\,\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{(k+1)^2}. \tag{8.3.30}$$

So, it just remains to check that $a_1 + b_1 \leq c$. Since $t_1 = 1$, $a_1 = \frac{2}{L} v_1$, while $b_1 = \|\boldsymbol{x}_1 - \boldsymbol{x}_\star\|_2^2$. In Lemma 8.3.2, set $\boldsymbol{x} = \boldsymbol{x}_\star$, $\boldsymbol{p} = \boldsymbol{p}_1$, to obtain

$$F(\boldsymbol{x}_\star) - F(\boldsymbol{x}_1) \geq \frac{L}{2} \|\boldsymbol{x}_1 - \boldsymbol{p}_1\|_2^2 + L \langle \boldsymbol{p}_1 - \boldsymbol{x}_\star, \boldsymbol{x}_1 - \boldsymbol{p}_1 \rangle \quad (8.3.31)$$

$$= \frac{L}{2} \left(\|\boldsymbol{x}_1 - \boldsymbol{x}_\star\|_2^2 - \|\boldsymbol{p}_1 - \boldsymbol{x}_\star\|_2^2 \right). \quad (8.3.32)$$

Since $\|\boldsymbol{p}_1 - \boldsymbol{x}_\star\|_2^2 = \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2$, this gives the result. □

8.4 Augmented Lagrange Multipliers

So far, we have described how to solve certain classes of *unconstrained* convex optimization problems arising in structured signal recovery. However, in some scenarios – e.g., if the noise level is low, or if the target solution $\boldsymbol{x}$ is known ahead of time to possess additional application-specific structure – it may be desirable to exactly enforce equality constraints such as in the exact BP program (8.1.1) or the PCP (8.1.5).

In this section, we describe a framework for solving *equality constrained* problems of the form

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}), \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \end{array} \quad (8.4.1)$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a convex function, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a matrix and $\boldsymbol{y} \in \text{range}(\boldsymbol{A})$ (so that the problem is feasible). One very intuitive approach to producing an approximate solution to (8.4.1) is to simply replace the inequality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ with a penalty function $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$, and solve the unconstrained problem

$$\text{minimize} \;\; g(\boldsymbol{x}) + \tfrac{\mu}{2} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \quad (8.4.2)$$

for a very large value of μ. As μ increases to $+\infty$, the solution set of this problem approaches the solution set of equality constrained problem (8.4.1). This is known as the *penalty method* in the optimization literature, and has a long history, with many variants. Its main advantage is that it leaves us with a simpler unconstrained problem, to which we can directly apply scalable first order methods such as the proximal gradient methods of the previous two sections.

However, there is a serious drawback to this approach. For first order methods such as PG and APG, the rate of convergence is dictated by how quickly the gradient $\nabla(\mu f) = \mu \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$ can change from point-to-point, which is measured through the Lipschitz constant

$$L_{\nabla \mu f} = \mu \|\boldsymbol{A}\|_2^2 .$$

This increases linearly with μ: *The larger μ is, the harder the unconstrained problem* (8.4.2) *is to solve!* One practical approach to mitigating this effect

is to solve a sequence of unconstrained problems, with increasing μ, and use the solution to each as an initial guess for the next. This *continuation* technique is often valuable in practice. Nevertheless, it suffers from the same drawback: as μ increases, accurate solutions become increasingly difficult to obtain.

Lagrange duality gives a more principled mechanism for studying and solving the constrained problem (8.4.1) via unconstrained problems. In particular, it will give us a mechanism for exactly solving the constrained problem (8.4.1) by solving a sequence of unconstrained problems *whose difficulty does not increase*. The central object in Lagrange duality is the Lagrangian

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \doteq g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle \,, \tag{8.4.3}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ is a vector of *Lagrange multipliers* corresponding to the equality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$. In particular, we can characterize optimal solutions $(\boldsymbol{x}, \boldsymbol{\lambda})$ as saddle points of the Lagrangian

$$\sup_{\boldsymbol{\lambda}} \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sup_{\boldsymbol{\lambda}} \inf_{\boldsymbol{x}} \; g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle \,. \tag{8.4.4}$$

If we define the dual function

$$d(\boldsymbol{\lambda}) \doteq \inf_{\boldsymbol{x}} \; g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle \,. \tag{8.4.5}$$

The saddle point characterization of optimal solutions suggests a natural computational approach to finding $(\boldsymbol{x}, \boldsymbol{\lambda})$:

$$\boldsymbol{x}_{k+1} \;=\; \arg\min_{\boldsymbol{x}} \; \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_k), \tag{8.4.6}$$

$$\boldsymbol{\lambda}_{k+1} \;=\; \boldsymbol{\lambda}_k + t_{k+1}(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}). \tag{8.4.7}$$

It is not difficult to show that $\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}$ is a subgradient[5] of the dual function $d(\boldsymbol{\lambda})$. This iteration corresponds to a subgradient ascent algorithm for maximizing the dual function, and hence is called *dual ascent*. In (8.4.7), t_{k+1} is the step size. For certain problem classes, dual ascent yields efficient, convergent algorithms, which produce an optimal primal-dual pair $(\boldsymbol{x}_\star, \boldsymbol{\lambda}_\star)$. However, for problems arising in structured signal recovery, the straightforward iteration (8.4.6)-(8.4.7) may fail:

Example 8.4.1. *Show that*

$$\inf_{\boldsymbol{x}} \; \|\boldsymbol{x}\|_1 + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle \;=\; \begin{cases} -\infty & \|\boldsymbol{A}^*\boldsymbol{\lambda}\|_\infty > 1, \\ -\langle \boldsymbol{\lambda}, \boldsymbol{y} \rangle & \|\boldsymbol{A}^*\boldsymbol{\lambda}\|_\infty \leq 1. \end{cases} \tag{8.4.8}$$

So, for basis pursuit, if Dual Ascent happens to produce a $\boldsymbol{\lambda}$ such that $\|\boldsymbol{A}^*\boldsymbol{\lambda}\|_\infty > 1$, the algorithm will break down. Notice, in particular, that when $\|\boldsymbol{A}^*\boldsymbol{\lambda}\|_\infty > 1$, we can produce arbitrarily large (in magnitude)

[5] Strictly, a *supergradient* since the dual $d(\boldsymbol{\lambda})$ is concave.

negative values of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ *by choosing* $\boldsymbol{x}$ *far away from the feasible set* $\{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}$.

This bad behavior occurs more generally. Thus, for structured signal recovery, the classical Lagrangian is sufficient for characterizing optimality conditions, but it does not penalize the equality $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ strongly enough for (8.4.6)-(8.4.7) to lead to a useful algorithm. A natural remedy is to *augment* the Lagrangian with an extra penalty term, by introducing the function

$$\mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{\lambda}) \doteq g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle + \frac{\mu}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 . \tag{8.4.9}$$

This function is known as the *Augmented Lagrangian* [Hestenes, 1969, Rockafellar, 1973, Powell, 1969]. As before, $\mu > 0$ is a penalty parameter. The augmented Lagrangian can be regarded as the Lagrangian for the constrained problem

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}) + \frac{\mu}{2} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array} \tag{8.4.10}$$

Since the penalty term $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$ is zero for all feasible $\boldsymbol{x}$, the optimal solutions of this problem coincide with the optimal solutions of the original problem (8.4.1).

Despite this formal equivalence, augmentation has dramatic consequences for numerical optimization. In particular, it can render the dual ascent iteration provably convergent, under very weak assumptions on the objective function g. To achieve this, we apply dual ascent to the penalized problem (8.4.10), and make a very particular choice of step size, $t_k = \mu$, yielding the iteration

$$\boldsymbol{x}_{k+1} \in \arg\min_{\boldsymbol{x}} \mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{\lambda}_k), \tag{8.4.11}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \mu\,(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}). \tag{8.4.12}$$

This iteration, with the particular choice $t_k = \mu$ is known as the *Method of Multipliers*. The update step (8.4.11) for $\boldsymbol{x}$ is a convex optimization problem itself and can typically be solved via the proximal gradient methods introduced in the previous sections.

Remark 8.4.2. *The choice* $t_k = \mu$ *is important, because it allows us to avoid the breakdown described in Example 8.4.1: To see this, since* $\boldsymbol{x}_{k+1}$ *minimizes the convex function* $\mathcal{L}_\mu$,

$$\begin{aligned} \boldsymbol{0} &\in \partial \mathcal{L}_\mu(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_k), \\ &= \partial g(\boldsymbol{x}_{k+1}) + \boldsymbol{A}^*\boldsymbol{\lambda}_k + \mu \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}), \\ &= \partial g(\boldsymbol{x}_{k+1}) + \boldsymbol{A}^*\boldsymbol{\lambda}_{k+1}, \\ &= \partial \mathcal{L}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}). \end{aligned}$$

Thus, $\boldsymbol{x}_{k+1}$ *minimizes the unaugmented Lagrangian* $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_{k+1})$ *with* $\boldsymbol{\lambda} = \boldsymbol{\lambda}_{k+1}$ *fixed. This means that* $d(\boldsymbol{\lambda}_{k+1}) > -\infty$, *and* $\boldsymbol{\lambda}_{k+1}$ *is* dual feasible *for*

the original problem. In particular, $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_{k+1})$ is bounded below. Because $\boldsymbol{\lambda}_{k+1}$ is always dual feasible, the bad behavior in Example 8.4.1 cannot occur.

Under appropriate assumptions on g, the iterates $\boldsymbol{x}_k$ produced by this modified algorithm converge to an optimal solution $\boldsymbol{x}_\star$ to the constrained problem (8.4.1). We state a slightly more general result that allows the penalty parameters μ to vary from iteration to iteration, as long as they remain bounded away from zero:

Theorem 8.4.3 (Convergence of Augmented Lagrangian)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a convex, coercive function, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ an arbitrary matrix, and $\boldsymbol{y} \in$ range$(\boldsymbol{A})$. Then the problem*

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \end{array} \tag{8.4.13}$$

has at least one optimal solution. Moreover, the ALM iteration

$$\boldsymbol{x}_{k+1} \in \arg\min_{\boldsymbol{x}} \mathcal{L}_{\mu_k}(\boldsymbol{x}, \boldsymbol{\lambda}_k), \tag{8.4.14}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \mu_k (\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}). \tag{8.4.15}$$

with sequence (μ_k) bounded away from zero produces a sequence $\boldsymbol{\lambda}_k$ that converges to a dual optimal solution. Moreover, every limit point of the sequence $\boldsymbol{x}_k$ is optimal for (8.4.13).

Figure 8.4 summarizes our general observations on the ALM method up to this point.

Remark 8.4.4 (More general convergence theorems)**.** *The statement of Theorem 8.4.3 represents a deliberate tradeoff between simplicity and generality. With somewhat more technical analysis, it is possible to show convergence ALM for more much general classes of g. The most practically important extension allows g to be an extended real valued function (a function from $\mathbb{R}^n$ to $\mathbb{R} \cup \{+\infty\}$). For example, if we wish to optimize a real-valued convex function g_0 over the set of $\boldsymbol{x}$ that satisfy the equality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$,* and *reside in some additional (nonempty closed, convex) constraint set C:*

$$\begin{array}{ll} \text{minimize} & g_0(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y},\ \boldsymbol{x} \in C, \end{array} \tag{8.4.16}$$

we can apply ALM to the problem

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}) \doteq g_0(\boldsymbol{x}) + \mathcal{I}_{\boldsymbol{x}\in C} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \end{array} \tag{8.4.17}$$

where $\mathcal{I}_{\boldsymbol{x}\in C}$ is the indicator function for C:

$$\boldsymbol{I}_{\boldsymbol{x}\in C} = \begin{cases} 0 & \boldsymbol{x} \in C \\ +\infty & \boldsymbol{x} \notin C. \end{cases} \tag{8.4.18}$$

Augmented Lagrange Multipler (ALM)

Problem Class:

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array}$$

$g : \mathbb{R}^n \to \mathbb{R}$ convex, $\boldsymbol{y} \in \text{range}(\boldsymbol{A})$.

Basic Iteration: set

$$\mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{\lambda}) = g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \rangle + \tfrac{\mu}{2} \left\| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \right\|_2^2 .$$

Repeat:

$$\begin{aligned} \boldsymbol{x}_{k+1} &\in \arg\min_{\boldsymbol{x}} \ \mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{\lambda}_k), \\ \boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \mu \left(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y} \right). \end{aligned}$$

Convergence Guarantee:

If g is coercive, every limit point of $(\boldsymbol{x}_k)$ is optimal.

Figure 8.4. Properties of the Augmented Lagrangian Method of Multipliers.

The survey of Eckstein [Eckstein, 2012] and the monograph of Bertsekas [Bertsekas, 1982b] are good introductory points for the more general theory, which enables such modifications.

Implementation Considerations.

The most important practical consideration is how to choose the sequence of penalty parameters $\mu^{(k)}$. As discussed above, this choice induces a trade-off between the cost of solving subproblems and the overall number of outer iterations – larger μ leaves us with fewer outer iterations, but harder subproblems. A typical strategy is to increase μ geometrically, up to some pre-fixed ceiling:

$$\mu_k = \min \left\{ \beta \mu_k, \mu_{\max} \right\},$$

where $\beta \approx 1.25$ is typical. The ceiling $\mu_{\max}$ is strongly problem dependent; choosing it "optimally" is something of a black art.

Our description and analysis of ALM assume that each of the subproblems is solved exactly. However, practically speaking, it may not be necessary to obtain high-accuracy solutions to the subproblems, especially in the early iterations. This can be justified theoretically. The choice of iterative method for solving the unconstrained subproblems is largely problem dependent. However, because the penalty term is quadratic, for many prob-

lems of interest in this text, the subproblems have composite form, and the APG algorithm applies.

In using APG (or any other iterative solver) to solve the unconstrained subproblems, it is highly advisible to use the previous iterate $\boldsymbol{x}_k$ as an initialization to solve for the subsequent iterate $\boldsymbol{x}_{k+1}$. While the subproblems are convex, and the global optimality of iterative algorithms does not depend on initialization, choosing an appropriate initializer can drastically reduce the overall number of iterations.

8.4.1 ALM for Basis Pursuit

We may apply ALM to the exact BP problem (8.1.1), which we summarize as Algorithm 8.5. This algorithm was introduced by [Yin et al., 2008], where it was interpreted as a *Bregman iteration*.

Algorithm 8.5 Augmented Lagrange Multipler (ALM) for BP

Problem: $\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1$ subj. to $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, given $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
Input: $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $\beta > 1$.
while $\{\boldsymbol{x}_k, \boldsymbol{\lambda}_k\}$ not converged $(k = 1, 2, \ldots)$ **do**
  $\boldsymbol{x}_{k+1} \leftarrow \arg\min \mathcal{L}_{\mu_k}(\boldsymbol{x}, \boldsymbol{\lambda}_k)$ using APG.
  $\boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \mu_k(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y})$.
  $\mu_{k+1} \leftarrow \min\{\beta\mu_k, \mu_{\max}\}$.
end while
Output: $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

8.4.2 ALM for Principal Component Pursuit

We recall principal component pursuit (PCP) solves the following program:

$$\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 \quad \text{subj. to} \quad \boldsymbol{Y} = \boldsymbol{L} + \boldsymbol{S}. \tag{8.4.19}$$

First, we rewrite the above program as a standard ALM objective function:

$$\mathcal{L}_\mu(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{\Lambda}) \;\doteq\; \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \langle \boldsymbol{\Lambda}, \boldsymbol{Y} - \boldsymbol{L} - \boldsymbol{S}\rangle + \tfrac{\mu}{2}\|\boldsymbol{Y} - \boldsymbol{L} - \boldsymbol{S}\|_F^2,$$

where $\mathcal{L}_\mu(\cdot)$ consists of a Lagrangian term with a Lagrange multiplier matrix $\boldsymbol{\Lambda}$ of the same size as $\boldsymbol{Y}$ and an augmented quadratic term that encourages the equality condition $\boldsymbol{Y} = \boldsymbol{L} + \boldsymbol{S}$.

Algorithm 8.6 Augmented Lagrange Multipler (ALM) for PCP

Problem: $\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1$ subj to $\boldsymbol{Y} = \boldsymbol{L} + \boldsymbol{S}$.
Input: $\boldsymbol{L}_0, \boldsymbol{S}_0 \in \mathbb{R}^{m\times n}$ and $\beta > 1$.
while $\{\boldsymbol{L}_k, \boldsymbol{S}_k\}$ not converged ($k = 1, 2, \ldots$). **do**
  $\{\boldsymbol{L}_{k+1}, \boldsymbol{S}_{k+1}\} \leftarrow \arg\min \mathcal{L}_{\mu_k}(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{\Lambda}_k)$ using APG.
  $\boldsymbol{\Lambda}_{k+1} \leftarrow \boldsymbol{\Lambda}_k + \mu_k(\boldsymbol{Y} - \boldsymbol{L}_{k+1} - \boldsymbol{S}_{k+1})$.
  $\mu_{k+1} \leftarrow \min\{\beta\mu_k, \mu_{\max}\}$.
end while
Output: $\boldsymbol{L}_\star \leftarrow \boldsymbol{L}_k, \boldsymbol{S}_\star \leftarrow \boldsymbol{S}_k$.

8.4.3 Convergence of ALM

In this subsection, we prove Theorem 8.4.3. The proof will actually reveal another interpretation of the Augmented Lagrangian method, as an application of the proximal point algorithm to the dual problem.

Proof. Let $d(\boldsymbol{\lambda})$ denote the dual function

$$d(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}} g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\rangle . \tag{8.4.20}$$

The dual function is concave, and so its negative

$$q(\boldsymbol{\lambda}) = -d(\boldsymbol{\lambda}) \tag{8.4.21}$$

is convex.

Note that for any $\boldsymbol{\lambda}$,

$$\begin{aligned} d(\boldsymbol{\lambda}) &\leq g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle \\ &= g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{\lambda}_{k+1}, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle + \langle \boldsymbol{\lambda} - \boldsymbol{\lambda}_{k+1}, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle . \end{aligned} \tag{8.4.22}$$

Now recall from Remark 8.4.2 that the augmented Lagragian method ensures that $\boldsymbol{x}_{k+1}$ minimizes the unaugmented Lagrangian $g(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\rangle$ with $\boldsymbol{\lambda} = \boldsymbol{\lambda}_{k+1}$ fixed. Hence, by definition of the function $d(\boldsymbol{\lambda})$, we have $d(\boldsymbol{\lambda}_{k+1}) = g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{\lambda}_{k+1}, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle$. Applying this to the above inequality, we obtain

$$d(\boldsymbol{\lambda}) \leq d(\boldsymbol{\lambda}_{k+1}) + \langle \boldsymbol{\lambda} - \boldsymbol{\lambda}_{k+1}, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle . \tag{8.4.23}$$

As $q(\boldsymbol{\lambda}) = -d(\boldsymbol{\lambda})$, we have

$$q(\boldsymbol{\lambda}) \geq q(\boldsymbol{\lambda}_{k+1}) + \langle \boldsymbol{\lambda} - \boldsymbol{\lambda}_{k+1}, \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{k+1}\rangle . \tag{8.4.24}$$

Hence $\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{k+1}$ is in the subgradient of $q(\cdot)$ at $\boldsymbol{\lambda}_{k+1}$, and

$$\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1} = \mu_k(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{k+1}) \in \mu_k \partial q(\boldsymbol{\lambda}_{k+1}), \tag{8.4.25}$$

and so

$$\boldsymbol{\lambda}_{k+1} = \mathrm{prox}_{\mu_k q}(\boldsymbol{\lambda}_k). \tag{8.4.26}$$

Thus, dual ascent corresponds to the proximal point iteration applied to $q(\cdot)$. Under our assumptions, strong duality obtains, the dual optimal value

$\sup_{\boldsymbol{\lambda}} d(\boldsymbol{\lambda}) > -\infty$ is finite, and a dual optimal solution $\bar{\boldsymbol{\lambda}}$ exists. Proposition 8.2.6 then implies that $\boldsymbol{\lambda}_k \to \boldsymbol{\lambda}_\star$, where $\boldsymbol{\lambda}_\star$ is some dual optimal point. This and the fact that μ_k is bounded away from zero give that

$$\|\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}\|_2 = \frac{\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k-1}\|_2}{\mu_k} \to 0, \tag{8.4.27}$$

and so the sequence $(\boldsymbol{x}_k)$ approaches the feasible set. From coercivity of g, there exists at least one primal optimal solution $\boldsymbol{x}_\star$. By optimality of $\boldsymbol{x}_{k+1}$, we have

$$g(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{\lambda}_k, \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\rangle + \frac{\mu}{2}\|\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y}\|_2^2 \;\le\; g(\boldsymbol{x}_\star). \tag{8.4.28}$$

For any cluster point $\bar{\boldsymbol{x}}$, continuity of g and $\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y} \to \boldsymbol{0}$ imply that $g(\bar{\boldsymbol{x}}) \le g(\boldsymbol{x}_\star)$, whence $g(\bar{\boldsymbol{x}}) = g(\boldsymbol{x}_\star)$. Hence, every cluster point is optimal. □

8.5 Alternating Direction Method of Multipliers

The previous section showed how augmented Lagrangian techniques could be used to solve equality constrained convex optimization problems, by reducing them to a sequence of unconstained subproblems. These subproblems may still be challenging optimization problems if we need to minimize against all the variables simultaneously. In many situations, though, it is possible to exploit special separable structures of the objective function and to alleviate the difficulty by reducing the overall optimization to multiple parallel subproblems of smaller sizes. We give two prominent examples:

Example 8.5.1 (Principal Component Pursuit)**.** *We solve*

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 \\ \text{subject to} & \boldsymbol{L} + \boldsymbol{S} = \boldsymbol{Y}. \end{array} \tag{8.5.1}$$

The objective function is separable *into two terms, $\|\cdot\|_*$ and $\|\cdot\|_1$, each of which has an efficient proximal operator.*

Example 8.5.2 (Distributed Learning)**.** *Special structure also arises in many large scale learning problems, where the goal is to fit a parametric model to a collection of observation vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$. Typically, we are provided with a* loss *$L(\boldsymbol{y}, \boldsymbol{x})$, which could be, e.g., the negative logarithm of the likelihood of observation $\boldsymbol{y}$ given parameters $\boldsymbol{x}$, and our goal is to minimize $\sum_i L(\boldsymbol{y}_i, \boldsymbol{x})$ over $\boldsymbol{x}$.*

In very large scale applications, it may be prohibitively expensive to store the $\boldsymbol{y}_i$ centrally, or to transmit them during the operation of an iterative algorithm. Rather, we can assume that they are stored in a distributed fashion, in N locations: the j-th location stores $\boldsymbol{y}_i$, $i \in I_j$. The loss on this

subset is $f_j(\boldsymbol{x}) = \sum_{i \in I_j} L(\boldsymbol{y}_i, \boldsymbol{x})$. *Our overall goal is then to solve*

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(\boldsymbol{x}). \tag{8.5.2}$$

Again, this objective function appears to separate into independent terms. To exploit this structure, we can introduce N *additional parameter vectors* $\boldsymbol{x}_i$*, which are constrained to coincide with* $\boldsymbol{x}$*:*

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) \\ \text{subject to} & \boldsymbol{x}_i = \boldsymbol{x} \quad i = 1, \ldots, N. \end{array} \tag{8.5.3}$$

In this section, we study a family of augmented Lagrangian algorithms that can exploit this special, separable structure. We begin by treating a generic problem of the form

$$\begin{array}{ll} \text{minimize} & g(\boldsymbol{x}) + h(\boldsymbol{z}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{y}, \end{array} \tag{8.5.4}$$

where g and h are convex functions, $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices, and $\boldsymbol{y} \in \text{range}([\boldsymbol{A} \mid \boldsymbol{B}])$. The Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda})$ associated with this problem simply is:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = g(\boldsymbol{x}) + h(\boldsymbol{z}) + \langle \boldsymbol{\lambda}, \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} \rangle . \tag{8.5.5}$$

As in the previous section, we form the augmented Lagrangian $\mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda})$ associated with this problem:

$$\mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = g(\boldsymbol{x}) + h(\boldsymbol{z}) + \langle \boldsymbol{\lambda}, \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} \rangle + \frac{\mu}{2} \left\| \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} \right\|_2^2 .$$

In many applications, including the examples listed above, it is easy to minimize $\mathcal{L}_\mu$ with respect to $\boldsymbol{x}$, when $\boldsymbol{\lambda}$ and $\boldsymbol{z}$ are fixed, and also easy to minimize it with respect to $\boldsymbol{z}$ when $\boldsymbol{\lambda}$ and $\boldsymbol{x}$ are fixed. This suggests a simple, alternating iteration

$$\boldsymbol{z}_{k+1} \in \arg\min_{\boldsymbol{z}} \mathcal{L}_\mu(\boldsymbol{x}_k, \boldsymbol{z}, \boldsymbol{\lambda}_k), \tag{8.5.6}$$

$$\boldsymbol{x}_{k+1} \in \arg\min_{\boldsymbol{x}} \mathcal{L}_\mu(\boldsymbol{x}, \boldsymbol{z}_{k+1}, \boldsymbol{\lambda}_k), \tag{8.5.7}$$

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \mu \left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{B}\boldsymbol{z}_{k+1} \right). \tag{8.5.8}$$

This is known as the *alternating directions method of multipliers*. In the numerical analysis literature, this style of updating is referred to as a *Gauss-Seidel iteration*. We recommend [Boyd et al., 2011a] for a friendly introduction to these methods, as well as useful recommendations on stopping criteria, parameter setting, etc.

8.5.1 ADMM for Principal Component Pursuit

When applied to the principal component pursuit program, the ADMM iteration takes on a particularly simple form. Here, the two groups of variables are the unknown low-rank matrix $\boldsymbol{L}$ and the unknown sparse error $\boldsymbol{S}$. The augmented Lagrangian is

$$\mathcal{L}_\mu(\boldsymbol{L},\boldsymbol{S},\boldsymbol{\Lambda}) = \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \langle\boldsymbol{\Lambda},\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}\rangle + \frac{\mu}{2}\|\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}\|_F^2. \tag{8.5.9}$$

The ADMM iteration sequentially updates $\boldsymbol{L}$, then $\boldsymbol{S}$, then $\boldsymbol{\Lambda}$. Each of these updates has a very simple familiar form. For example,

$$\begin{aligned}
\boldsymbol{L}_{k+1} &= \arg\min_{\boldsymbol{L}} \mathcal{L}_\mu(\boldsymbol{L},\boldsymbol{S}_k,\boldsymbol{\Lambda}_k)\\
&= \arg\min_{\boldsymbol{L}} \|\boldsymbol{L}\|_* + \langle\boldsymbol{\Lambda}_k,\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}_k\rangle + \frac{\mu}{2}\|\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}_k\|_F^2\\
&= \arg\min_{\boldsymbol{L}} \|\boldsymbol{L}\|_* + \frac{\mu}{2}\left\|\boldsymbol{Y}-\boldsymbol{S}_k+\mu^{-1}\boldsymbol{\Lambda}_k-\boldsymbol{L}\right\|_F^2 + \varphi(\boldsymbol{S}_k,\boldsymbol{\Lambda}_k)\\
&= \operatorname{prox}_{\mu^{-1}\|\cdot\|_*}\left(\boldsymbol{Y}-\boldsymbol{S}_k+\mu^{-1}\boldsymbol{\Lambda}_k\right).
\end{aligned} \tag{8.5.10}$$

Thus, the update step for the low-rank term can be evaluated simply by computing the proximal operator for the nuclear norm.

A similar simple rule can be derived for the sparse term:

$$\begin{aligned}
\boldsymbol{S}_{k+1} &= \arg\min_{\boldsymbol{S}} \mathcal{L}_\mu(\boldsymbol{L}_{k+1},\boldsymbol{S},\boldsymbol{\Lambda}_k)\\
&= \arg\min_{\boldsymbol{S}} \lambda\|\boldsymbol{S}\|_1 + \langle\boldsymbol{\Lambda}_k,\boldsymbol{Y}-\boldsymbol{L}_{k+1}-\boldsymbol{S}\rangle + \frac{\mu}{2}\|\boldsymbol{Y}-\boldsymbol{L}_{k+1}-\boldsymbol{S}\|_F^2\\
&= \arg\min_{\boldsymbol{S}} \lambda\|\boldsymbol{S}\|_1 + \frac{\mu}{2}\left\|\boldsymbol{Y}-\boldsymbol{L}_{k+1}+\mu^{-1}\boldsymbol{\Lambda}_k-\boldsymbol{S}\right\|_F^2 + \varphi(\boldsymbol{L}_{k+1},\boldsymbol{\Lambda}_k)\\
&= \operatorname{prox}_{\lambda\mu^{-1}\|\cdot\|_1}\left(\boldsymbol{Y}-\boldsymbol{L}_{k+1}+\mu^{-1}\boldsymbol{\Lambda}_k\right).
\end{aligned} \tag{8.5.11}$$

Combining these two observations, we obtain a simple, lightweight algorithm for solving the Principal Component Pursuit program.

Algorithm 8.7 ADMM for Principal Component Pursuit

Problem: $\min_{\boldsymbol{L},\boldsymbol{S}} \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \langle\boldsymbol{\Lambda},\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}\rangle + \frac{\mu}{2}\|\boldsymbol{Y}-\boldsymbol{L}-\boldsymbol{S}\|_F^2$, given $\boldsymbol{Y}$, $\lambda,\mu>0$.
Input: $\boldsymbol{L}_0 \in \mathbb{R}^{m\times n}$, $\boldsymbol{S}_0 \in \mathbb{R}^{m\times n}$.
while $((\boldsymbol{L}_k,\boldsymbol{S}_k)$ not converged $(k=1,2,\ldots)$ **do**
  $\boldsymbol{L}_{k+1} \leftarrow \operatorname{prox}_{\mu^{-1}\|\cdot\|_*}\left(\boldsymbol{Y}-\boldsymbol{S}_k+\mu^{-1}\boldsymbol{\Lambda}_k\right)$.
  $\boldsymbol{S}_{k+1} \leftarrow \operatorname{prox}_{\lambda\mu^{-1}\|\cdot\|_1}\left(\boldsymbol{Y}-\boldsymbol{L}_{k+1}+\mu^{-1}\boldsymbol{\Lambda}_k\right)$.
  $\boldsymbol{\Lambda}_{k+1} \leftarrow \boldsymbol{\Lambda}_k + \mu(\boldsymbol{Y}-\boldsymbol{L}_{k+1}-\boldsymbol{S}_{k+1})$.
end while
Output: $\boldsymbol{L}_\star \leftarrow \boldsymbol{L}_k; \boldsymbol{S}_\star \leftarrow \boldsymbol{S}_k$.

8.5.2 Convergence of ADMM

There has been a rich history and literature on charactering the convergence and convergence rates of the ADMM algorithm under various conditions [Deng and Yin, 2016]. The ADMM can be naturally viewed as an approximation to the classical ALM method studied in the previous section: In the case the objective function is separable, one uses a single pass of "Gauss-Seidel" block minimization to substitute for full minimization of the augmented Lagrangian in each iteration (8.4.11). However, as pointed out in [Eckstein, 2012], this interpretation does not seem to lead to any known convergence proof for the ADMM that is closely related the convergence proof for ALM given in Section 8.4.3. Proofs for strong convergence of ADMM under general assumptions and for many of its variants are considerably more involved than that for ALM.

In this section, we try to establish a convergence result of the ADMM algorithm for a very basic case that nevertheless is applicable to problems of interest to us, such as the principal component pursuit. Our proof follows the approach of [Deng and Yin, 2016]. In our analysis below, we will use the following monotonicity property of convex functions.

Lemma 8.5.3 (Monotonicity Property)**.** *Given a convex function* $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *and any* $\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^n$ *such that* $\boldsymbol{v} \in \partial f(\boldsymbol{x})$ *and* $\boldsymbol{v}' \in \partial f(\boldsymbol{x}')$*, we have*

$$\langle \boldsymbol{x} - \boldsymbol{x}', \boldsymbol{v} - \boldsymbol{v}' \rangle \geq 0. \tag{8.5.12}$$

Proof. From the definition of subgradient, we have

$$f(\boldsymbol{x}') \geq f(\boldsymbol{x}) + \langle \boldsymbol{v}, \boldsymbol{x}' - \boldsymbol{x} \rangle, \quad f(\boldsymbol{x}) \geq f(\boldsymbol{x}') + \langle \boldsymbol{v}', \boldsymbol{x} - \boldsymbol{x}' \rangle. \tag{8.5.13}$$

Adding these two inequalities together we obtain:

$$f(\boldsymbol{x}) + f(\boldsymbol{x}') \geq f(\boldsymbol{x}) + f(\boldsymbol{x}') + \langle \boldsymbol{v} - \boldsymbol{v}', \boldsymbol{x}' - \boldsymbol{x} \rangle. \tag{8.5.14}$$

Canceling $f(\boldsymbol{x}) + f(\boldsymbol{x}')$ from both sides obtains the desired result. □

To simplify the proof of convergence, we will assume our problem satisfies all necessary nice properties. For instance, the functions $g(\boldsymbol{x})$ and $h(\boldsymbol{z})$ are both convex and have Lipschitz continuous gradient with constant L. We also assume that the problem (8.5.4) has a unique solution $(\boldsymbol{x}_\star, \boldsymbol{z}_\star, \boldsymbol{\lambda}_\star)$ that satisfies the KKT conditions, or in other words, a saddle point of the Lagrangian (8.5.5):

$$\boldsymbol{A}^* \boldsymbol{\lambda}_\star \in \partial g(\boldsymbol{x}_\star), \quad \boldsymbol{B}^* \boldsymbol{\lambda}_\star \in \partial h(\boldsymbol{z}_\star), \quad \boldsymbol{A}\boldsymbol{x}_\star + \boldsymbol{B}\boldsymbol{z}_\star - \boldsymbol{y} = 0. \tag{8.5.15}$$

Theorem 8.5.4. *Under the above assumptions, the ADMM iterates* (8.5.7) *to* (8.5.8) *converge to the KKT solution* $(\boldsymbol{x}_\star, \boldsymbol{z}_\star, \boldsymbol{\lambda}_\star)$ *in the following sense:*

$$\boldsymbol{\lambda}_k \to \boldsymbol{\lambda}_\star, \quad \boldsymbol{A}\boldsymbol{x}_k \to \boldsymbol{A}\boldsymbol{x}_\star, \quad \boldsymbol{B}\boldsymbol{z}_k \to \boldsymbol{B}\boldsymbol{z}_\star. \tag{8.5.16}$$

In the case when $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *both have full column rank, we have* $\boldsymbol{x}_k \to \boldsymbol{x}_\star$ *and* $\boldsymbol{z}_k \to \boldsymbol{z}_\star$.

Proof. Recall from (8.5.8) that we have

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \mu\left(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{B}\boldsymbol{z}_{k+1}\right). \tag{8.5.17}$$

Combining this with the conditions for the KKT solution (8.5.15), we have

$$\boldsymbol{A}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star) + \boldsymbol{B}(\boldsymbol{z}_{k+1} - \boldsymbol{z}_\star) = \frac{1}{\mu}(\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}). \tag{8.5.18}$$

From the optimality of the subproblems (8.5.7) and (8.5.6), we have

$$\begin{aligned} \boldsymbol{A}^*\boldsymbol{\lambda}_{k+1} &\in \partial g(\boldsymbol{x}_{k+1}), & (8.5.19)\\ \boldsymbol{B}^*\big(\boldsymbol{\lambda}_{k+1} - \mu\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})\big) &\in \partial h(\boldsymbol{z}_{k+1}). & (8.5.20) \end{aligned}$$

Applying the Lemma 8.5.3 the iterates $(\boldsymbol{x}_{k+1}, \boldsymbol{z}_{k+1})$ and the KKT solution $(\boldsymbol{x}_\star, \boldsymbol{z}_\star)$, we get:

$$\begin{aligned} \langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_\star, \boldsymbol{A}^*(\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_\star)\rangle &\geq 0, & (8.5.21)\\ \langle \boldsymbol{z}_{k+1} - \boldsymbol{z}_\star, \boldsymbol{B}^*\big(\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_\star - \mu\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})\big)\rangle &\geq 0. & (8.5.22) \end{aligned}$$

Adding the above two inequalities and using (8.5.18), we get

$$\begin{aligned} & \frac{1}{\mu}\langle \boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_\star, \boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\rangle + \langle \boldsymbol{x}_{k+1} - \boldsymbol{x}_\star, \mu\boldsymbol{A}^*\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})\rangle \\ \geq\ & \langle \boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1}), \boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\rangle \geq 0. \end{aligned}$$

In the last inequality, we have used the fact that $\boldsymbol{A}^*\boldsymbol{\lambda}_k$ is the subgradient of $g(\boldsymbol{x})$ at $\boldsymbol{x}_k$ from equation (8.5.19) and the monotonicity Lemma 8.5.3. Define $\|\boldsymbol{x}\|_{\boldsymbol{Q}}^2 = \boldsymbol{x}^*\boldsymbol{Q}\boldsymbol{x}$ and the above inequality leads to:

$$\begin{aligned} & \frac{1}{\mu}\langle \boldsymbol{\lambda}_k - \boldsymbol{\lambda}_\star, \boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\rangle + \langle \boldsymbol{x}_k - \boldsymbol{x}_\star, \mu\boldsymbol{A}^*\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})\rangle \\ \geq\ & \frac{1}{\mu}\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\|^2 + \mu\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\|_{\boldsymbol{A}^*\boldsymbol{A}}^2. \end{aligned}$$

Notice that we have the identity

$$\|\boldsymbol{a} - \boldsymbol{c}\|_{\boldsymbol{Q}}^2 - \|\boldsymbol{b} - \boldsymbol{c}\|_{\boldsymbol{Q}}^2 = 2(\boldsymbol{a} - \boldsymbol{c})^*\boldsymbol{Q}(\boldsymbol{a} - \boldsymbol{b}) - \|\boldsymbol{a} - \boldsymbol{b}\|_{\boldsymbol{Q}}^2.$$

Let $\boldsymbol{a}$ be the k-th iterate, $\boldsymbol{b}$ be the $(k+1)$-th iterate, and $\boldsymbol{c}$ be the KKT solution. Substitute the left side of the inequality into the right side of this identity, we obtain:

$$\begin{aligned} & \frac{1}{\mu}\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_\star\|^2 + \mu\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_{\boldsymbol{A}^*\boldsymbol{A}}^2 - \big(\frac{1}{\mu}\|\boldsymbol{\lambda}_{k+1} + \boldsymbol{\lambda}_\star\|^2 - \mu\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star\|_{\boldsymbol{A}^*\boldsymbol{A}}^2\big) \\ \geq\ & \frac{1}{\mu}\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\|^2 + \mu\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\|_{\boldsymbol{A}^*\boldsymbol{A}}^2. \end{aligned}$$

From the above inequality, we see that the sequence

$$\frac{1}{\mu}\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_\star\|^2 + \mu\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|^2_{\boldsymbol{A}^*\boldsymbol{A}}$$

is monotonically non-increasing and thus must converge. This implies that the right hand side must converge to zero:

$$\frac{1}{\mu}\|\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1}\|^2 + \mu\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\|^2_{\boldsymbol{A}^*\boldsymbol{A}} \to 0. \tag{8.5.23}$$

Therefore, $\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k+1} \to 0$ and $\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{A}\boldsymbol{x}_{k+1} \to 0$.

It is easy to check that any converging subsequence of $(\boldsymbol{x}_k, \boldsymbol{z}_k, \boldsymbol{\lambda}_k)$ converges to a point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}}, \bar{\boldsymbol{\lambda}})$ that satisfies the KKT condition. As the solution that satisfies the KKT condition is unique, we have

$$\boldsymbol{\lambda}_k \to \boldsymbol{\lambda}_\star, \quad \boldsymbol{A}\boldsymbol{x}_k \to \boldsymbol{A}\boldsymbol{x}_\star, \quad \boldsymbol{B}\boldsymbol{z}_k \to \boldsymbol{B}\boldsymbol{z}_\star.$$

Finally, when the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ both have full column rank, such as the case with principal component pursuit, we have

$$\boldsymbol{x}_k \to \boldsymbol{x}_\star, \quad \boldsymbol{z}_k \to \boldsymbol{z}_\star.$$

□

The above theorem is not the most general and strongest result that one can obtain for the convergence of ADMM. We recommend [Eckstein, 2012] for a tutorial introduction to a more formal convergence analysis of ADMM. For more recent analysis of generalized version of ADMM including their convergence rates, we recommend the work of [Deng and Yin, 2016]. Like the ALM algorithm in the previous section, the most natural approach is to recognize ADMM as some known algorithm applied to the dual. In fact, ADMM turns out to be equivalent to Douglas-Rachford splitting applied to the dual. For more details on this, see [Eckstein and Bertsekas, 1992, Combettes and Wajs, 2005]. ADMM has also been widely applied to problems when the number of terms are more than three. The convergence analysis of ADMM with more than three terms is much more difficult and it has been shown to diverge in many cases.

8.6 Frank-Wolfe Methods for Scalable Optimization

In the previous sections, we showed how the special structure of optimization problems arising in sparse and low-dimensional data analysis can be leveraged to obtain efficient and scalable algorithms. One key piece of structure was the existence of an easy-to-compute proximal operator. For example, for nuclear norm minimization we showed that at a point

$\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$,

$$\operatorname{prox}_{\gamma\|\cdot\|_*}(\boldsymbol{Z}) = \boldsymbol{U}\mathcal{S}_\lambda[\boldsymbol{\Sigma}]\boldsymbol{V}^*, \tag{8.6.1}$$

where $\mathcal{S}_\lambda[\cdot]$ is the soft thresholding operator. Using the proximal operator, we obtain proximal gradient methods that enjoy the same convergence rate as if the objective were smooth, even though it is nonsmooth. Each iteration consists of simple linear operations, followed by the application of $\operatorname{prox}_{\gamma\|\cdot\|_*}$. Each iteration can be computed in time polynomial in the size of the target matrix: the proximal operator can be computed in time $O(n_1 n_2 \max\{n_1, n_2\})$ in the worst case. This is sufficient for moderate-sized datasets where n_1 and n_2 are each in the thousands.

However, many problems in data science and imaging require even more scalable solutions. In this section, we introduce a classical method from optimization, known as the *Frank-Wolfe* or *conditional gradient* algorithm, which is scalable enough to solve extremely large sparse and low-rank recovery problems. The key property of this method is that in each iteration, it solves a subproblem which is simpler and easier to compute than the proximal operator.

In its classical form, the Frank-Wolfe algorithm applies to the problem of optimizing a smooth, convex function over a compact convex set:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}), \\ \text{subject to} & \boldsymbol{x} \in C. \end{array} \tag{8.6.2}$$

Here, the objective function f is assumed to be a convex, differentiable function whose gradient $\nabla f(\boldsymbol{x})$ L-Lipschitz. The constraint set C is assumed to be a closed, bounded convex set with diameter

$$\operatorname{diam}(C) \doteq \max\left\{\|\boldsymbol{x} - \boldsymbol{x}'\|_2 \mid \boldsymbol{x}, \boldsymbol{x}' \in C\right\}. \tag{8.6.3}$$

Constrained formulations of sparse and low-rank recovery.

Many of the sparse and low-rank recovery problems that we have considered thus far can be reformulated in terms of (8.6.2). For example, for sparse recovery, we can choose $C = \{\boldsymbol{x} \mid \|\boldsymbol{x}\|_1 \leq \tau\}$ to be an ℓ^1 ball, and solve

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2, \\ \text{subject to} & \|\boldsymbol{x}\|_1 \leq \tau. \end{array} \tag{8.6.4}$$

Similarly, for low-rank matrix completion, we can choose C to be a nuclear norm ball and solve

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\mathcal{P}_\Omega[\boldsymbol{X}] - \boldsymbol{Y}\|_F^2, \\ \text{subject to} & \|\boldsymbol{X}\|_* \leq \tau. \end{array} \tag{8.6.5}$$

Exercises 8.6 – 8.7 explore further reformulations of unconstrained sparse and low-rank optimization in the form (8.6.2).

The Frank-Wolfe Method.

Similar to the other methods we have discussed thus far, Frank-Wolfe is an iterative method, which generates a sequence of iterates $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \ldots$ as follows. At each iteration, we generate a new point $\boldsymbol{v}_k$ by solving a constrained optimization problem

$$\boldsymbol{v}_k \in \arg\min_{\boldsymbol{v} \in C} \langle \boldsymbol{v}, \nabla f(\boldsymbol{x}_k) \rangle . \tag{8.6.6}$$

We then set

$$\boldsymbol{x}_{k+1} = (1 - \gamma_k)\boldsymbol{x}_k + \gamma_k \boldsymbol{v}_k \in C, \tag{8.6.7}$$

where $\gamma_k \in (0, 1)$ is a specially chosen step size. Figure 8.5 summarizes the properties of this method.

Interpretation as Minimizing a First-Order Approximation.

The Frank-Wolfe method can be interpreted as follows. At a given point $\boldsymbol{x}_k$, we form a first order approximation to the objective function f:

$$f(\boldsymbol{v}) \approx \hat{f}(\boldsymbol{v}, \boldsymbol{x}_k) \doteq f(\boldsymbol{x}_k) + \langle \boldsymbol{v} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k) \rangle . \tag{8.6.8}$$

We minimize the approximation $\hat{f}(\boldsymbol{v}, \boldsymbol{x}_k)$ over $\boldsymbol{v} \in C$ to produce $\boldsymbol{v}_k$. We then take a step in the direction $\boldsymbol{w}_k = \boldsymbol{v}_k - \boldsymbol{x}_k$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \gamma_k \boldsymbol{w}_k. \tag{8.6.9}$$

Computing the Step Direction.

The crucial subproblem in the Frank-Wolfe method involves minimizing a linear function over a compact convex set C:

$$\min_{\boldsymbol{v} \in C} \langle \boldsymbol{v}, \nabla f(\boldsymbol{x}) \rangle . \tag{8.6.10}$$

Depending on the constraint set C, this could itself be a challenging (or even intractable!) optimization problem. Fortunately, for the problems of interest in this book, this subproblem can be solved in an efficient and scalable manner. We give two examples below:

Example 8.6.1 (Frank-Wolfe Subproblem over an ℓ^1 Ball)**.** *Consider the problem*

$$\min_{\|\boldsymbol{v}\|_1 \leq \tau} \langle \boldsymbol{v}, \boldsymbol{g} \rangle . \tag{8.6.11}$$

Let i be any index for which $\boldsymbol{g}_i = \|\boldsymbol{g}\|_\infty$, and $\sigma_i = \operatorname{sign}(\boldsymbol{g}_i)$. Then (8.6.11) has a solution

$$\boldsymbol{v}_\star = -\tau \sigma_i \boldsymbol{e}_i, \tag{8.6.12}$$

where $\boldsymbol{e}_i$ is the i-th standard basis vector. The solution $\boldsymbol{v}_\star$ can be computed in linear time, simply by finding the largest magnitude entry of $\boldsymbol{g}$.

Example 8.6.2 (Frank-Wolfe Subproblem of a Nuclear Norm Ball)**.** *Consider the problem*

$$\min_{\|\boldsymbol{V}\|_* \leq \tau} \langle \boldsymbol{V}, \boldsymbol{G} \rangle . \tag{8.6.13}$$

Let $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^* = \sum_{i=1}^{n_1} \boldsymbol{u}_i \sigma_i \boldsymbol{v}_i$ *denote the singular value decomposition of* $\boldsymbol{G}$*. Then* (8.6.13) *has an optimal solution*

$$\boldsymbol{V}_\star = -\tau \boldsymbol{u}_1 \boldsymbol{v}_1^*. \tag{8.6.14}$$

This optimal solution can be computed in time $O(n_1 n_2)$ *by computing (only) the lead singular vector pair* $(\boldsymbol{u}_1, \boldsymbol{v}_1)$ *of* $\boldsymbol{G}$*.*

The latter example illustrates the special virtue of the Frank-Wolfe method in nuclear norm minimization: the key subproblem only requires us to compute *one* singular value/vector triple. For a problem involving $n_1 \times n_2$ matrices, this can be done in time $O(n_1 n_2)$ – a dramatic improvement over proximal gradient methods, which require a full singular value decomposition in each iteration.[6]

This scalability comes at a price, though. Compared to accelerated proximal gradient methods, which converge at a rate of $O(1/k^2)$ in function values, the Frank-Wolfe method only achieves a rate of $O(1/k)$. The following theorem gives a precise bound on the worst case rate of convergence for Frank-Wolfe over the class of convex functions with Lipschitz gradient.

Theorem 8.6.3. *Let* $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ *denote the sequence of iterates generated by the Frank-Wolfe method, with step size* $\gamma_k = \frac{2}{k+2}$*. Then*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star) \leq \frac{2L\mathrm{diam}^2(C)}{k+2}. \tag{8.6.15}$$

We prove this theorem in the next subsection. In the context of nuclear norm minimization, this result can be viewed as follows: Frank-Wolfe allows us to derive methods that produce moderate-quality solutions to extremely large problems, for which methods with better worst cases rates simply take too long to compute even a single iterate.

8.6.1 Convergence of Frank-Wolfe

In this section, we prove Theorem 8.6.3 on the convergence rate of the Frank-Wolfe method.

Proof. For ease of notation, write $d = \mathrm{diam}(C)^2$, $\boldsymbol{x} = \boldsymbol{x}_k$, $\boldsymbol{x}^+ = \boldsymbol{x}_{k+1}$, $\gamma = \gamma_k$, and $\boldsymbol{v} = \boldsymbol{v}_k$. Note that

$$\boldsymbol{x}^+ - \boldsymbol{x} = \gamma (\boldsymbol{v} - \boldsymbol{x}) . \tag{8.6.16}$$

[6] See Exercise **??** for a different attempt to address this issue in proximal gradient methods.

Frank-Wolfe Method (FW)

Problem Class:

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

$f : \mathbb{R}^n \to \mathbb{R}$ convex, differentiable, $\nabla f(\boldsymbol{x})$ L-Lipschitz.
C a compact convex set.

Basic Iteration: Repeat

$$\begin{aligned}\boldsymbol{v}_k &\in \arg\min_{\boldsymbol{v}\in C} \langle \boldsymbol{v}, \nabla f(\boldsymbol{x}_k)\rangle,\\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \gamma_k(\boldsymbol{v}_k - \boldsymbol{x}_k),\end{aligned}$$

with $\gamma_k = \frac{2}{k+2}$.

Convergence Guarantee:

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star) \le \frac{2L\,\mathrm{diam}^2(C)}{k+2}.$$

Figure 8.5. Properties of the Frank-Wolfe Method.

Because $\nabla f(\boldsymbol{x})$ is L-Lipschitz, we can use the upper bound (8.2.8) to obtain

$$\begin{aligned} f(\boldsymbol{x}^+) &\le f(\boldsymbol{x}) + \left\langle \nabla f(\boldsymbol{x}), \boldsymbol{x}^+ - \boldsymbol{x}\right\rangle + \tfrac{L}{2}\left\|\boldsymbol{x}^+ - \boldsymbol{x}\right\|_2^2\\ &\le f(\boldsymbol{x}) + \gamma\left\langle \nabla f(\boldsymbol{x}), \boldsymbol{v} - \boldsymbol{x}\right\rangle + \tfrac{\gamma^2 L}{2}\left\|\boldsymbol{v} - \boldsymbol{x}\right\|_2^2\\ &\le f(\boldsymbol{x}) + \gamma\left\langle \nabla f(\boldsymbol{x}), \boldsymbol{v} - \boldsymbol{x}\right\rangle + \tfrac{\gamma^2 L}{2} d^2. \end{aligned} \tag{8.6.17}$$

Meanwhile, by convexity,

$$\begin{aligned} f(\boldsymbol{x}_\star) &\ge f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}_\star - \boldsymbol{x}\rangle &(8.6.18)\\ &\ge f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{v} - \boldsymbol{x}\rangle, &(8.6.19)\end{aligned}$$

where the final line follows because $\boldsymbol{v}$ is chosen to minimize $\langle \nabla f(\boldsymbol{x}), \boldsymbol{v}\rangle$. Combining these two inequalities, we find that

$$\langle \nabla f(\boldsymbol{x}), \boldsymbol{v} - \boldsymbol{x}\rangle \le -\Big(f(\boldsymbol{x}) - f(\boldsymbol{x}_\star)\Big), \tag{8.6.20}$$

whence, plugging into (8.6.17) and subtracting $f(\boldsymbol{x}_\star)$ from both sides, we obtain

$$f(\boldsymbol{x}^+) - f(\boldsymbol{x}_\star) \le (1-\gamma)\Big(f(\boldsymbol{x}) - f(\boldsymbol{x}_\star)\Big) + \tfrac{\gamma^2}{2} L d^2. \tag{8.6.21}$$

We use this basic relationship together with an inductive argument to bound the rate of convergence of the Frank-Wolfe method. Let ε_k denote

the suboptimality (in function values) at iteration k:

$$\varepsilon_k = f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star). \tag{8.6.22}$$

Set $\gamma_k = \frac{2}{k+2}$, so $\gamma_0 = 1$. Applying (8.6.21), we find that

$$\varepsilon_1 \leq \tfrac{1}{2} L d^2. \tag{8.6.23}$$

Suppose now that for $\ell = 1, \ldots, k$, $\varepsilon_\ell \leq \frac{2}{\ell+2} L d^2$. Applying (8.6.21) again, we find that

$$\begin{aligned} \varepsilon_{k+1} &\leq \frac{k}{k+2}\varepsilon_k + \frac{2}{(k+2)^2} L d^2 \\ &\leq \frac{k+1}{(k+2)^2} \times 2 L d^2 \\ &\leq \frac{2 L d^2}{(k+1)+2}. \end{aligned} \tag{8.6.24}$$

Hence, the relationship $\varepsilon_\ell \leq \frac{2}{\ell+2} L d^2$ holds for all iterations ℓ, as claimed. □

8.6.2 Frank-Wolfe for Stable Matrix Completion

We illustrate the general Frank-Wolfe method for the particular problem of recovering a low-rank matrix from incomplete and noisy observations

$$\boldsymbol{Y} = \mathcal{P}_\Omega[\boldsymbol{X}_{\text{true}} + \boldsymbol{Z}], \tag{8.6.25}$$

where $\boldsymbol{X}_{\text{true}} \in \mathbb{R}^{n_1 \times n_2}$ has low rank, $\boldsymbol{Z} \in \mathbb{R}^{n_1 \times n_2}$ is a matrix of small, dense noise, and $\Omega \subseteq [n_1] \times [n_2]$ is the set of observed entries. One approach to approximately recovering $\boldsymbol{X}_{\text{true}}$ is to minimize the reconstruction error over the set of all matrices of small nuclear norm:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{X}) \equiv \frac{1}{2} \left\| \mathcal{P}_\Omega[\boldsymbol{X}] - \boldsymbol{Y} \right\|_F^2, \\ \text{subject to} & \|\boldsymbol{X}\|_* \leq \tau. \end{array} \tag{8.6.26}$$

Here, the constraint $\|\boldsymbol{X}\|_* \leq \tau$ encourages $\boldsymbol{X}$ to have low rank. The constraint set $C = \{\boldsymbol{X} \mid \|\boldsymbol{X}\|_* \leq \tau\}$ is closed and bounded. Moreover, the gradient

$$\nabla f(\boldsymbol{X}) = \mathcal{P}_\Omega\left[\boldsymbol{X} - \boldsymbol{Y}\right] \tag{8.6.27}$$

is 1-Lipschitz, and so the Frank-Wolfe method indeed applies to this problem.

The key step in the Frank-Wolfe method is to minimize a linear function $\langle \boldsymbol{V}, \nabla f(\boldsymbol{X}) \rangle$ over the constraint set C. As described above, this problem can be solved in closed form: if

$$\nabla f(\boldsymbol{X}) = \sum_{i=1}^{n_1} \boldsymbol{u}_i \sigma_i \boldsymbol{v}_i^* \tag{8.6.28}$$

is the singular value decomposition of f, then

$$-\tau \boldsymbol{u}_1 \boldsymbol{v}_1 \in \arg\min_{\boldsymbol{V} \in C} \langle \boldsymbol{V}, \nabla f(\boldsymbol{X}) \rangle \,. \tag{8.6.29}$$

The lead singular value / vectors can be extracted from the matrix $\nabla f(\boldsymbol{X})$ efficiently, without computing the entire SVD (8.6.28). Typically, this is done using the power method, which was described in some detail in Chapter 4 and Exercise 4.6. To cleanly describe the method, we simply let

$$(\boldsymbol{u}_1, \sigma_1, \boldsymbol{v}_1) = \text{LeadSV}(\boldsymbol{G}) \tag{8.6.30}$$

denote the operation which extracts a leading singular value / vector triple from a matrix $\boldsymbol{G}$. Using this notation, the complete Frank-Wolfe algorithm for stable matrix completion is describe in Algorithm 8.8.

Algorithm 8.8 Frank-Wolfe for Stable Matrix Completion

Problem: $\min_{\boldsymbol{X}} \frac{1}{2} \|\mathcal{P}_\Omega[\boldsymbol{X}] - \boldsymbol{Y}\|_F^2$ subj. to $\|\boldsymbol{X}\|_* \leq \tau$, given $\boldsymbol{Y} \in \mathbb{R}^{n_1 \times n_2}$ and $\Omega \subseteq [n_1] \times [n_2]$.
Input: $\boldsymbol{X}_0 \in \mathbb{R}^{n_1 \times n_2}$ satisfying $\|\boldsymbol{X}_0\|_* \leq \tau$.
while $\boldsymbol{X}_k$ not converged $(k = 1, 2, \ldots)$ **do**
  $(\boldsymbol{u}_1, \sigma_1, \boldsymbol{v}_1) \leftarrow \text{LeadSV}\left(\mathcal{P}_\Omega\left[\boldsymbol{X}_k - \boldsymbol{Y}\right]\right)$.
  $\boldsymbol{V}_k \leftarrow -\tau \boldsymbol{u}_1 \boldsymbol{v}_1^*$.
  $\boldsymbol{X}_{k+1} \leftarrow \frac{k}{k+2} \boldsymbol{X}_k + \frac{2}{k+2} \boldsymbol{V}_k$.
end while
Output: $\boldsymbol{X}_\star \leftarrow \boldsymbol{X}_k$.

The Frank-Wolfe method requires only a single singular value/vector triple at each iteration. Moreover, since $\boldsymbol{V}_k = -\tau \boldsymbol{u}_1 \boldsymbol{v}_1^*$ has rank one, the rank of $\boldsymbol{X}_k$ increases by at most one at each iteration. In this sense, Frank-Wolfe can be viewed as a *greedy method*. It constructs a low-rank matrix $\boldsymbol{X}_\star$ by adding on one (optimally chosen) rank-one factor at a time.

8.6.3 Connection to Greedy Methods for Sparsity

In sparse and low-rank approximation, *greedy methods* are sometimes favored for their simplicity and scalability. For sparse approximation, the Frank-Wolfe method gives one such greedy algorithm. Consider the problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \equiv \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \,, \\ \text{subject to} & \|\boldsymbol{x}\|_1 \leq \tau. \end{array} \tag{8.6.31}$$

Notice that

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}). \tag{8.6.32}$$

The Frank-Wolfe subproblem

$$\min_{\|\boldsymbol{v}\|_1 \leq \tau} \langle \boldsymbol{v}, \nabla f(\boldsymbol{x}) \rangle \tag{8.6.33}$$

has an especially simple solution: letting i be the index of the largest magnitude entry of ∇f, and σ its sign,

$$\boldsymbol{v}_\star = -\tau \sigma \boldsymbol{e}_i. \tag{8.6.34}$$

Algorithm 8.9 Frank-Wolfe for Noisy Sparse Recovery

Problem: $\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ subj. to $\|\boldsymbol{x}\|_1 \leq \tau$, given $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
Input: $\boldsymbol{x}_0 \in \mathbb{R}^n$ satisfying $\|\boldsymbol{x}_0\|_1 \leq \tau$.
while $\boldsymbol{x}_k$ not converged $(k = 1, 2, \ldots)$ **do**
  $i_k \leftarrow \arg\max_i |[\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})]_i|$.
  $\sigma \leftarrow \operatorname{sign}\left([\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})]_{i_k}\right)$.
  $\boldsymbol{v}_k \leftarrow -\tau \sigma \boldsymbol{e}_{i_k}$.
  $\boldsymbol{x}_{k+1} \leftarrow \frac{k}{k+2} \boldsymbol{x}_k + \frac{2}{k+2} \boldsymbol{v}_k$.
end while
Output: $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

Algorithm 8.9 describes in detail the Frank-Wolfe method for problem (8.6.31). At each iteration, it increases the number of nonzero entries in the vector $\boldsymbol{x}$ by at most one, by adding on a multiple of $\boldsymbol{e}_{i_k}$. Let

$$I_k = \{i_1, \ldots, i_{k-1}\} = \operatorname{supp}(\boldsymbol{x}_k) \tag{8.6.35}$$

denote the collection of indices that have been choosen up to time k. We generate I_{k+1} from I_k by introducing a (potentially) new index

$$I_{k+1} = I_k \cup \{i_k\}. \tag{8.6.36}$$

This new index is chosen according to the largest-magnitude entry in the gradient ∇f. Write

$$\boldsymbol{A} = [\boldsymbol{a}_1 \mid \cdots \mid \boldsymbol{a}_n] \tag{8.6.37}$$

for the columns of $\boldsymbol{A}$, and let

$$\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y} \tag{8.6.38}$$

denote the measurement residual at point $\boldsymbol{x}_k$. Since $\nabla f(\boldsymbol{x}_k) = \boldsymbol{A}^* \boldsymbol{r}_k$, the Frank-Wolfe method chooses the index i_k corresponding to the column $\boldsymbol{a}_{i_k}$ that is most correlated with the residual $\boldsymbol{r}_k$.

Matching Pursuit.

A number of classical *greedy methods* for sparse approximation have this basic structure. A canonical example is the *Matching Pursuit* algorithm.

Algorithm 8.10 Matching Pursuit for Sparse Approximation

Problem: find a sparse $\boldsymbol{x}$ such that $f(\boldsymbol{x}) \equiv \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ is small.
$\boldsymbol{x}_0 \leftarrow \boldsymbol{0}$.
while $\boldsymbol{x}_k$ not converged $(k = 1, 2, \ldots)$ **do**
  $i_k \leftarrow \arg\max_i |[\boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})]_i|$.
  $t_k \leftarrow -\frac{\langle \boldsymbol{a}_{i_k}, \boldsymbol{r}_k \rangle}{\|\boldsymbol{a}_{i_k}\|_2^2}$.
  $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + t_k \boldsymbol{e}_{i_k}$.
end while
Output: $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

This algorithm generates a sequence of iterates $\boldsymbol{x}_0 = \boldsymbol{0}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, by repeatedly choosing a column of $\boldsymbol{a}_{i_k}$ of $\boldsymbol{A}$ that is most correlated with the residual $\boldsymbol{r}_k$. Similar to Frank-Wolfe, Matching Pursuit[7] sets

$$i_k = \arg\max_i |[\nabla f(\boldsymbol{x}_k)]_i| = \arg\max_i |\boldsymbol{a}_i^* \boldsymbol{r}_k| \,. \tag{8.6.39}$$

However, rather than stepping a predetermined length in the $\boldsymbol{e}_{i_k}$ direction, Matching Pursuit chooses the step size t_k by solving a one-dimensional minimization problem:

$$t_k \;=\; \arg\min_t f\left(\boldsymbol{x}_k + t_k \boldsymbol{e}_{i_k}\right) \;=\; -\frac{\langle \boldsymbol{a}_{i_k}, \boldsymbol{r}_k \rangle}{\|\boldsymbol{a}_{i_k}\|_2^2}. \tag{8.6.40}$$

This can be viewed as a form of *exact line search* and typically leads to more rapid convergence in practice. The overall Matching Pursuit algorithm is specified as Algorithm 8.10.

Orthogonal Matching Pursuit.

Matching pursuit achieves better convergence by choosing the step size t_k in an optimal manner. Since

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_k \boldsymbol{e}_{i_k}, \tag{8.6.41}$$

this is equivalent to making an optimal choice of the i_k-th entry in $\boldsymbol{x}_{k+1}$, while leaving all of the other entries fixed. It is possible to further improve the rate of convergence of this approach by choosing *all* of the nonzero entries of $\boldsymbol{x}_{k+1}$ optimally (rather than just the i_k-th entry). In notation, let $I_k = \{i_1, i_2, \ldots, i_{k-1}\}$ denote the collection of indices that have been chosen up to step k. The *Orthogonal Matching Pursuit* method selects an

[7] Despite the strong parallels to the Frank-Wolfe method, Matching Pursuit was originally motivated from completely different principles. Its introduction to the literature on sparse approximation predates the use of the Frank-Wolfe method; see the Notes and References to this chapter for more historical details and connections to classical greedy algorithms in other fields.

Algorithm 8.11 Orthogonal Matching Pursuit for Sparse Approximation

Problem: find a sparse $\boldsymbol{x}$ such that $f(\boldsymbol{x}) \equiv \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ is small.
$\boldsymbol{x}_0 \leftarrow \boldsymbol{0}$, $I_0 \leftarrow \varnothing$.
while $\boldsymbol{x}_k$ not converged ($k = 1, 2, \ldots$) **do**
  $\boldsymbol{r}_k \leftarrow \boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}$.
  $i_k \leftarrow \arg\max_i |\boldsymbol{a}_i^* \boldsymbol{r}_k|$.
  $I_{k+1} \leftarrow I_k \cup \{i_k\}$.
  $[\boldsymbol{x}_{k+1}]_{I_{k+1}} \leftarrow \left(\boldsymbol{A}_{I_{k+1}}^* \boldsymbol{A}_{I_{k+1}}\right)^{-1} \boldsymbol{A}_{I_{k+1}}^* \boldsymbol{y}$.
  $[\boldsymbol{x}_{k+1}]_{I_{k+1}^c} \leftarrow \boldsymbol{0}$.
end while
Output: $\boldsymbol{x}_\star \leftarrow \boldsymbol{x}_k$.

index i_k that maximizes the correlation $|\boldsymbol{a}_i^* \boldsymbol{r}_k|$ of a column of $\boldsymbol{A}$ with the residual $\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}$. It sets $I_{k+1} = I_k \cup \{i_k\}$, and then updates all of the nonzero entries in $\boldsymbol{x}$ by setting

$$\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}} \ \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \quad \text{subject to} \quad \operatorname{supp}(\boldsymbol{x}) \subseteq I_{k+1}. \tag{8.6.42}$$

This problem can be solved in closed form:

$$[\boldsymbol{x}_{k+1}]_{I_{k+1}} = \left(\boldsymbol{A}_{I_{k+1}}^* \boldsymbol{A}_{I_{k+1}}\right)^{-1} \boldsymbol{A}_{I_{k+1}}^* \boldsymbol{y}, \tag{8.6.43}$$

$$[\boldsymbol{x}_{k+1}]_{I_{k+1}^c} = \boldsymbol{0}. \tag{8.6.44}$$

The name *Orthogonal* Matching Pursuit comes from the observation that the residual

$$\begin{aligned} \boldsymbol{r}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{y} \\ &= \left(\boldsymbol{A}_{I_{k+1}} \left(\boldsymbol{A}_{I_{k+1}}^* \boldsymbol{A}_{I_{k+1}}\right)^{-1} \boldsymbol{A}_{I_{k+1}}^* - \boldsymbol{I}\right) \boldsymbol{y} \end{aligned} \tag{8.6.45}$$

is orthogonal to the range $\operatorname{range}\left(\boldsymbol{A}_{I_{k+1}}\right)$ of the dictionary columns selected through the first $k+1$ iterations.

The overall Orthogonal Matching Pursuit algorithm is given as Algorithm 8.11. This method is sometimes favored by practitioners due to its simplicity, and the fact that it maintains an explicit *active set* I_k. The latter property is useful for problems in which the support of the sparse solution $\boldsymbol{x}_\star$ is the object of interest.[8]

Although OMP has many variants and extensions, it was originally derived for the specific problem of finding sparse near-solutions to a linear system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$. Like ℓ^1 minimization, OMP is guaranteed to succeed whenever $\boldsymbol{y}$ is generated from some sufficiently sparse $\boldsymbol{x}_o$ and the

[8] For example, in the RF spectrum sensing discussed in Chapter 12, the goal is to determine which bands of the RF spectrum are occupied, in order to avoid interference. The specific energy levels within these bands are of secondary importance.

columns of $\boldsymbol{A}$ are sufficiently spread in the high-dimensional space $\mathbb{R}^m$. In particular:

Theorem 8.6.4. *Suppose that* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$, *with*

$$k = \|\boldsymbol{x}_o\|_0 \leq \frac{1}{2\mu(\boldsymbol{A})}. \tag{8.6.46}$$

Then after k *iterations, the OMP algorithm terminates with* $\boldsymbol{x}_k = \boldsymbol{x}_o$ *and* $I_k = \operatorname{supp}(\boldsymbol{x}_o)$.

Exercise 8.8 guides the reader through a proof of Theorem 8.6.4. The key message of the proof is that under the conditions of the theorem, at each iteration ℓ the algorithm selects an index i_ℓ that belongs to the true support set $\operatorname{supp}(\boldsymbol{x}_o)$.

The form of Theorem 8.6.4 can be directly compared to that of Theorem 3.2.3 of Chapter 3. These results imply that *both* OMP and ℓ^1 minimization recover $\boldsymbol{x}_o$ whenever $\|\boldsymbol{x}_o\|_0 \leq 1/2\mu(\boldsymbol{A})$. Hence, at an intuitive level, both methods succeed whenever the target solution is sparse and the matrix $\boldsymbol{A}$ is "nice". However, as shown in Chapter 3, the incoherence condition requires $\boldsymbol{x}_o$ to be extremely sparse. ℓ^1 minimization also recovers denser $\boldsymbol{x}_o$ under the stronger condition that $\boldsymbol{A}$ satisfies the Restricted Isometry Property $\delta_{Ck}(\boldsymbol{A}) < c$. While various improved analyses of OMP are available, the RIP is not sufficient for OMP to succeed. In this sense, convex relaxation achieves a better uniform guarantee. However, OMP can be modified to also guarantee sparse recovery under the RIP. The key ideas are to allow the algorithm to *remove* elements from the active set I_k at each iteration, and to add multiple elements. The resulting method, called COSAMP (Compressed Sampling Matching Pursuit) is described in more detail in Exercise 8.9. The large and varied literature on greedy algorithms also includes greedy methods for more general problems such as low-rank recovery; see the Notes and References to this chapter.

8.7 Notes and References

The name *basis pursuit* was first suggested by Chen and Donoho in their early work on compressive sensing [Chen, 1995, Chen et al., 2001].

More references about the proximity operator [Moreau, 1962, Combettes and Wajs, 2005].

In the literature, ISTA has been also studied under different names, such as forward-backward splitting [Combettes and Wajs, 2005], thresholded Landweber [Daubechies et al., 2004], and separable approximation (SpaRSA) [Wright et al., 2008].

Review the history of ADMM algorithms.

There are several important heuristics in the implementation of convex optimization algorithms, which are significant enough that practitioners shall pay close attention.

8.8 Exercises

8.1 (Proximal Operator for the Nuclear Norm)**.** *Prove the first and the third assertions of Proposition 8.2.3.*

8.2 (Iterative Soft-Thresholding Algorithm for PCP)**.** *Regarding the solving the stable principal component pursuit program using proximal gradient descent:*

- *Apply the proximal gradient method to the PCP program. Based on separability of the two non-smooth terms in the objective function, write down the corresponding proximal operators and the associated* $\boldsymbol{w}_1$ *and* $\boldsymbol{w}_2$*. Justify the updates for* $\boldsymbol{L}_{k+1}$ *and* $\boldsymbol{S}_{k+1}$ *in Algorithm 8.7.*
- *Code a MATLAB function that implements the Iterative Soft-Thresholding Algorithm 8.7 for PCP.*

8.3 (Accelerated Proximal Gradient Algorithm for Robust PCA*)**.** *Code a MATLAB function that implement the accelerated proximal gradient method for the stable PCP program.*

8.4 (Dual Feasibility Condition of ALM)**.**

8.5 (Implementation: Augmented Lagrange Multiplier Algorithm for PCP)**.**

8.6 (Unconstrained problems via Frank-Wolfe)**.**

8.7 (Sparse and Low-Rank via Frank-Wolfe)**.** *... ...*

8.8 (Sparse recovery by Orthogonal Matching Pursuit)**.** *The goal of this exercise is to prove Theorem 8.6.4, which shows that OMP correctly recovers any target sparse solution* $\boldsymbol{x}_o$ *with* $k = \|\boldsymbol{x}_o\|_0 \leq \frac{1}{2\mu(\boldsymbol{A})}$*. Let* $I = \operatorname{supp}(\boldsymbol{x}_o)$*.*

- OMP selects a true support index in the first iteration. *Let* $i_{\max}$ *index a maximum-magnitude entry of* $\boldsymbol{x}_o$*, i.e.,* $\boldsymbol{x}_o(i_{\max}) = \|\boldsymbol{x}_o\|_\infty$*. Using the incoherence of* $\boldsymbol{A}$*, argue that*

$$\left|\boldsymbol{a}_{i_{\max}}^* \boldsymbol{r}_0\right| \geq \left|\boldsymbol{a}_j^* \boldsymbol{r}_0\right| \quad \forall\, j \in I^c. \tag{8.8.1}$$

- *Argue by induction that OMP selects some* $i_\ell \in I$ *for every iteration* $\ell = 0, \ldots, k-1$*.*
- *Using the fact that* $r_\ell \perp \operatorname{span}(\boldsymbol{A}_{I_\ell})$*, argue that OMP selects a* new *index* $i_\ell \in I$ *at each iteration* $\ell = 0, \ldots, k-1$*. Conclude that OMP terminates with* $\boldsymbol{x}_k = \boldsymbol{x}_o$ *and* $I_k = I$*, as claimed.*

8.9 (Greedy Methods that Succeed Under RIP)**.** *The Compressive Sampling Matching Pursuit (COSAMP) algorithm modifies OMP by adding and subtracting multiple indices from the active set I_ℓ at each iteration ℓ. This algorithm takes as input a target number of nonzero entries, s, and modifies OMP as follows:*

- *At each iteration ℓ, we let $I_{\ell+1/2} = I_\ell \cup$*
- *We then solve for $\boldsymbol{x}_{\ell+1/2}$ by*
- *We then*

Implement the COSAMP algorithm, and compare its breakdown in terms of sparsity level to OMP.

8.10 (ADMM for Basis Pursuit)**.**

8.11 (Dual of Principal Component Pursuit)**.** *Show that the dual program to the PCP program is*

$$\max_{\boldsymbol{\Lambda}} \operatorname{trace}(\boldsymbol{Y}^* \boldsymbol{\Lambda}) \quad \textit{subj. to} \quad J(\boldsymbol{\Lambda}) \leq 1,$$

where $\boldsymbol{\Lambda}$ is the matrix of Lagrange multipliers for the equality constraint $\boldsymbol{Y} = \boldsymbol{L} + \boldsymbol{S}$, and $J(\boldsymbol{\Lambda}) = \max(\|\boldsymbol{\Lambda}\|_2, \lambda^{-1}\|\boldsymbol{\Lambda}\|_\infty)$.