2
Optimization Algorithms: An Overview

Contents

In this book we are primarily interested in optimization algorithms, as opposed to "modeling," i.e., the formulation of real-world problems as mathematical optimization problems, or "theory," i.e., conditions for strong duality, optimality conditions, etc. In our treatment, we will mostly focus on guaranteeing convergence of algorithms to desired solutions, and the associated rate of convergence and complexity analysis. We will also discuss special characteristics of algorithms that make them suitable for particular types of large scale problem structures, and distributed (possibly asynchronous) computation. In this chapter we provide an overview of some broad classes of optimization algorithms, their underlying ideas, and their performance characteristics.

Iterative algorithms for minimizing a function $f : \Re^n \mapsto \Re$ over a set X generate a sequence $\{x_k\}$, which will hopefully converge to an optimal solution. In this book we focus on iterative algorithms for the case where X is convex, and f is either convex or is nonconvex but differentiable. Most of these algorithms involve one or both of the following two ideas, which will be discussed in Sections 2.1 and 2.2, respectively:

(a) *Iterative descent*, whereby the generated sequence $\{x_k\}$ is feasible, i.e., $\{x_k\} \subset X$, and satisfies

$$\phi(x_{k+1}) < \phi(x_k) \qquad \text{if and only if } x_k \text{ is not optimal,}$$

where ϕ is a *merit function*, that measures the progress of the algorithm towards optimality, and is minimized only at optimal points, i.e.,

$$\arg\min_{x \in X} \phi(x) = \arg\min_{x \in X} f(x).$$

Examples are $\phi(x) = f(x)$ and $\phi(x) = \inf_{x^* \in X^*} \|x - x^*\|$, where X^* is the set of optimal points, assumed nonempty. In some cases, iterative descent may be the primary idea, but modifications or approximations are introduced for a variety of reasons. For example one may modify an iterative descent method to make it suitable for distributed asynchronous computation, or to deal with random or nonrandom errors, but in the process lose the iterative descent property. In this case, the analysis is appropriately modified, but often maintains important aspects of its original descent-based character.

(b) *Approximation*, whereby the generated sequence $\{x_k\}$ need not be feasible, and is obtained by solving at each k an approximation to the original optimization problem, i.e.,

$$x_{k+1} \in \arg\min_{x \in X_k} F_k(x),$$

where F_k is a function that approximates f and X_k is a set that approximates X. These may depend on the prior iterates $x_0, \ldots, x_k$,

as well as other parameters. Key ideas here are that minimization of F_k over X_k should be easier than minimization of f over X, and that x_k should be a good starting point for obtaining x_{k+1} via some (possibly special purpose) method. Of course, the approximation of f by F_k and/or X by X_k should improve as k increases, and there should be some convergence guarantees as $k \to \infty$. We will summarize the main approximation ideas of this book in Section 2.2.

A major class of problems that we aim to solve is dual problems, which by their nature involve nondifferentiable optimization. The fundamental reason is that the negative of a dual function is typically a conjugate function, which is closed and convex, but need not be differentiable. Moreover nondifferentiable cost functions naturally arise in other contexts, such as exact penalty functions, and machine learning with ℓ_1 regularization. Accordingly many of the algorithms that we discuss in this book do not require cost function differentiability for their application.

Still, however, differentiability plays a major role in problem formulations and algorithms, so it is important to maintain a close connection between differentiable and nondifferentiable optimization approaches. Moreover, nondifferentiable problems can often be converted to differentiable ones by using a smoothing scheme (see Section 2.2.5). We consequently summarize in Section 2.1 some of the main ideas of iterative algorithms that rely on differentiability, such as gradient and Newton methods, and their incremental variants. We return to some of these ideas in Sections 6.1-6.3, but for most of the remainder of the book we focus primarily on convex possibly nondifferentiable cost functions.

Since the present chapter has an overview character, our discussion will not be supplemented by complete proofs; in many cases we will provide just intuitive explanations and refer to the literature for a more detailed analysis. In subsequent chapters we will treat various types of algorithms in greater detail. In particular, in Chapter 3, we discuss descent-type iterative methods that use subgradients. In Chapters 4 and 5, we discuss primarily the approximation approach, focusing on two types of algorithms and their combinations: polyhedral approximation and proximal, respectively. In Chapter 6, we discuss a number of additional methods, which extend and combine the ideas of the preceding chapters.

2.1 ITERATIVE DESCENT ALGORITHMS

Iterative algorithms generate sequences $\{x_k\}$ according to

$$x_{k+1} = G_k(x_k),$$

where $G_k : \Re^n \mapsto \Re^n$ is some function that may depend on k, and x_0 is some starting point. In a more general context, G_k may depend on

some preceding iterates $x_{k-1}, x_{k-2}, \ldots$. We are typically interested in the convergence of the generated sequence $\{x_k\}$ to some desirable point. We are also interested in questions of rate of convergence, such as for example the number of iterations needed to bring a measure of error to within a given tolerance, or asymptotic bounds on some measure of error as the number of iterations increases.

A *stationary* iterative algorithm is obtained when G_k does not depend on k, i.e.,

$$x_{k+1} = G(x_k).$$

This algorithm aims to solve a fixed point problem: finding a solution of the equation $x = G(x)$. A classical optimization example is the gradient iteration

$$x_{k+1} = x_k - \alpha \nabla f(x_k), \tag{2.1}$$

which aims at satisfying the optimality condition $\nabla f(x) = 0$ for an unconstrained minimum of a differentiable function $f : \Re^n \mapsto \Re$. Here α is a positive stepsize parameter that is used to ensure that the iteration makes progress towards the solution set of the corresponding problem. Another example is the iteration

$$x_{k+1} = x_k - \alpha(Qx_k - b) = (I - \alpha Q)x_k + \alpha b, \tag{2.2}$$

which aims at solution of the linear system $Qx = b$, where Q is a matrix that has eigenvalues with positive real parts (so that the matrix $I - \alpha Q$ has eigenvalues within the unit circle for sufficiently small $\alpha > 0$, and the iteration is convergent to the unique solution). If f is the quadratic function $f(x) = \frac{1}{2}x'Qx - b'x$, where Q is positive definite symmetric, then we have $\nabla f(x_k) = Qx_k - b$ and the gradient iteration (2.1) can be written in the form (2.2).

Convergence of the stationary iteration $x_{k+1} = G(x_k)$ can be ascertained in a number of ways. The most common is to verify that G is a *contraction mapping* with respect to some norm, i.e., for some $\rho < 1$, and some norm $\|\cdot\|$ (not necessarily the Euclidean norm), we have

$$\big\|G(x) - G(y)\big\| \leq \rho\|x - y\|, \qquad \forall\ x, y \in \Re^n.$$

Then it can be shown that G has a unique fixed point x^*, and $x_k \to x^*$, starting from any $x_0 \in \Re^n$; this is the well-known Banach Fixed Point Theorem (see Prop. A.4.1 in Section A.4 of Appendix A, where the contraction and other approaches for convergence analysis are discussed). An example is the mapping

$$G(x) = (I - \alpha Q)x + \alpha b$$

of the linear iteration (2.2), when the eigenvalues of $I - \alpha Q$ lie strictly within the unit circle.

The case where G is a contraction mapping provides an example of convergence analysis based on a descent approach: at each iteration we have

$$\|x_{k+1} - x^*\| \leq \rho \|x_k - x^*\|, \tag{2.3}$$

so the distance $\|x - x^*\|$ is decreased with each iteration at a nonsolution point x. Moreover, in this case we obtain an estimate of the convergence rate: $\|x_k - x^*\|$ is decreased at least as fast as the geometric progression $\{\rho^k \|x_0 - x^*\|\}$; this is called *linear* or *geometric* convergence.†

Many optimization algorithms involve a contraction mapping as described above. There are also other types of convergent fixed point iterations, which do not require that G is a contraction mapping. In particular, there are cases where G is a *nonexpansive mapping* [$\rho = 1$ in Eq. (2.3)], and there is sufficient structure in G to ensure a form of improvement of an appropriate figure of merit at each iteration; the proximal algorithm, introduced in Section 2.2.3 and discussed in detail in Chapter 5, is an important example of this type.

There are also many cases of *nonstationary* iterations of the form

$$x_{k+1} = G_k(x_k),$$

whose convergence analysis is difficult or impossible with a contraction or nonexpansive mapping approach. An example is unconstrained minimization of a differentiable function f with a gradient method of the form

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{2.4}$$

where the stepsize α_k is not constant. Still many of these algorithms admit a convergence analysis based on a descent approach, whereby we introduce a function ϕ that measures the progress of the algorithm towards optimality, and show that

$$\phi(x_{k+1}) < \phi(x_k) \qquad \text{if and only if } x_k \text{ is not optimal.}$$

Two common cases are when $\phi(x) = f(x)$ or $\phi(x) = \text{dist}(x, X^*)$, the Euclidean minimum distance of x from the set X^* of minima of f. For example convergence of the gradient algorithm (2.4) is often analyzed by showing that for all k,

$$f(x_{k+1}) \leq f(x_k) - \gamma_k \big\|\nabla f(x_k)\big\|^2,$$

where γ_k is a positive scalar that depends on α_k and some characteristics of f, and is such that $\sum_{k=0}^{\infty} \gamma_k = \infty$; this brings to bear the convergence

† Generally, we say that a nonnegative scalar sequence $\{\beta_k\}$ *converges (at least) linearly or geometrically* if there exist scalars $\gamma > 0$ and $\rho \in (0, 1)$ such that $\beta_k \leq \gamma \rho^k$ for all k. For a discussion of different definitions of linear and other types of convergence rate, see [OrR70], [Ber82a], and [Ber99].

methodology of Section A.4 in Appendix A and guarantees that either $\nabla f(x_k) \to 0$ or $f(x_k) \to -\infty$.

In what follows in this section we will provide an overview of iterative optimization algorithms that rely on some form of descent for their validity, we discuss some of their underlying motivation, and we raise various issues that will be discussed later. We will also provide in the exercises a sampling of some related convergence analysis, while deferring to subsequent chapters a more detailed theoretical development. Moreover, in the present section we focus in greater detail on the differentiable cost function case and the potential benefits of differentiability. Our focus in subsequent chapters will be primarily on nondifferentiable problems.

2.1.1 Differentiable Cost Function Descent – Unconstrained Problems

A natural iterative descent approach to minimizing a real-valued function $f : \Re^n \mapsto \Re$ over a set X is based on cost improvement: starting with a point $x_0 \in X$, construct a sequence $\{x_k\} \subset X$ such that

$$f(x_{k+1}) < f(x_k), \qquad k = 0, 1, \ldots,$$

unless x_k is optimal for some k, at which time the method stops.

In this context it is useful to consider the directional derivative of f at a point x in a direction d. For a differentiable f, it is given by

$$f'(x; d) = \lim_{\alpha \downarrow 0} \frac{f(x + \alpha d) - f(x)}{\alpha} = \nabla f(x)' d, \tag{2.5}$$

(cf. Section A.3 of Appendix A). From this formula it follows that if d_k is a *descent direction* at x_k, in the sense that

$$f'(x_k; d_k) < 0,$$

we may reduce the cost by moving from x_k along d_k with a small enough positive stepsize α. In the unconstrained case where $X = \Re^n$, this leads to an algorithm of the form

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2.6}$$

where d_k is a descent direction at x_k and α_k is a positive scalar stepsize. If no descent direction can be found at x_k, i.e., $f'(x_k; d) \geq 0$, for all $d \in \Re^n$, from Eq. (2.5) it follows that x_k must satisfy the necessary condition for optimality

$$\nabla f(x_k) = 0.$$

Gradient Methods for Differentiable Unconstrained Minimization

For the case where f is differentiable and $X = \Re^n$, there are many popular descent algorithms of the form (2.6). An important example is the classical gradient method, where we use $d_k = -\nabla f(x_k)$ in Eq. (2.6):

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k).$$

Since for differentiable f we have

$$f'(x_k; d) = \nabla f(x_k)' d,$$

it follows that

$$-\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} = \arg \min_{\|d\| \leq 1} f'(x_k; d)$$

[assuming $\nabla f(x_k) \neq 0$]. Thus the gradient method is the descent algorithm of the form (2.6) that uses the direction that yields the greatest rate of cost improvement. For this reason it is also called the method of *steepest descent*.

Let us now discuss the convergence rate of the steepest descent method, assuming that f is twice continuously differentiable. With proper stepsize choice, it can be shown that the method has a linear rate, assuming that it generates a sequence $\{x_k\}$ that converges to a vector x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. For example, if α_k is a sufficiently small constant $\alpha > 0$, the corresponding iteration

$$x_{k+1} = x_k - \alpha \nabla f(x_k), \tag{2.7}$$

can be shown to be contractive within a sphere centered at x^*, so it converges linearly.

To get a sense of this, assume for convenience that f is quadratic,† so by adding a suitable constant to f, we have

$$f(x) = \tfrac{1}{2}(x - x^*)' Q (x - x^*), \qquad \nabla f(x) = Q(x - x^*),$$

† Convergence analysis using a quadratic model is commonly used in nonlinear programming. The rationale is that behavior of an algorithm for a positive definite quadratic cost function is typically a correct predictor of its behavior for a twice differentiable cost function in the neighborhood of a minimum where the Hessian matrix is positive definite. Since the gradient is zero at that minimum, the positive definite quadratic term dominates the other terms in the Taylor series expansion, and the asymptotic behavior of the method does not depend on terms of order higher than two.

This time-honored line of analysis underlies some of the most widely used unconstrained optimization methods, such as Newton, quasi-Newton, and conjugate direction methods, which will be briefly discussed later. However, the rationale for these methods is weakened when the Hessian is singular at the minimum, since in this case third order terms may become significant. For this reason, when considering algorithmic options for a given differentiable optimization problem, it is important to consider (in addition to its cost function structure) whether the problem is "singular or "nonsingular."

where Q is the positive definite symmetric Hessian of f. Then for a constant stepsize α, the steepest descent iteration (2.7) can be written as

$$x_{k+1} - x^* = (I - \alpha Q)(x_k - x^*).$$

For $\alpha < 2/\lambda_{max}$, where λ_{max} is the largest eigenvalue of Q, the matrix $I - \alpha Q$ has eigenvalues strictly within the unit circle, and is a contraction with respect to the Euclidean norm. It can be shown (cf. Exercise 2.1) that the optimal modulus of contraction can be achieved with the stepsize choice

$$\alpha^* = \frac{2}{M + m},$$

where M and m are the minimum and maximum eigenvalues of Q. With this stepsize, we obtain the linear convergence rate estimate

$$\|x_{k+1} - x^*\| \leq \left(\frac{\frac{M}{m} - 1}{\frac{M}{m} + 1} \right) \|x_k - x^*\|. \tag{2.8}$$

Thus the convergence rate of steepest descent may be estimated in terms of the *condition number of* Q, the ratio M/m of largest to smallest eigenvalue. As the condition number increases to ∞ (i.e., the problem is increasingly "ill-conditioned") the modulus of contraction approaches 1, and the convergence can be very slow. This is the dominant characteristic of the behavior of gradient methods for the class of twice differentiable problems with positive definite Hessian. This class of problems is very broad, so condition number issues often become the principal consideration when implementing gradient methods in practice.

Choosing an appropriate constant stepsize may require some preliminary experimentation. Another possibility is the *line minimization rule*, which uses some specialized line search algorithm to determine

$$\alpha_k \in \arg\min_{\alpha \geq 0} f\big(x_k - \alpha \nabla f(x_k)\big).$$

With this rule, when the steepest descent method converges to a vector x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, its convergence rate is also linear, but not faster than the one of Eq. (2.8), which is associated with an optimally chosen constant stepsize (see [Ber99], Section 1.3).

If the method converges to an optimal point x^* where the Hessian matrix $\nabla^2 f(x^*)$ is singular or does not exist, the convergence rate that we can guarantee is typically slower than linear. For example, with a properly chosen constant stepsize, and under some reasonable conditions (Lipschitz continuity of ∇f), we can show that

$$f(x_k) - f^* \leq \frac{c(x_0)}{k}, \qquad k = 1, 2, \ldots, \tag{2.9}$$

where f^* is the optimal value of f and $c(x_0)$ is a constant that depends on the initial point x_0 (see Section 6.1).

For problems where ∇f is continuous but cannot be assumed Lipschitz continuous at or near the minimum, it is necessary to use a stepsize rule that can produce time-varying stepsizes. For example in the scalar case where $f(x) = |x|^{3/2}$, the steepest descent method with any constant stepsize oscillates around the minimum $x^* = 0$, because the gradient grows too fast around x^*. However, the line minimization rule as well as other rules, such as the Armijo rule to be discussed shortly, guarantee a satisfactory form of convergence (see the end-of-chapter exercises and the discussion of Section 6.1).

On the other hand, with additional assumptions on the structure of f, we can obtain a faster convergence than the $O(1/k)$ estimate on the cost function error of Eq. (2.9). In particular, the rate of convergence to a singular minimum depends on the order of growth of the cost function near that minimum; see [Dun81], which shows that if f is convex, has a unique minimum x^*, and satisfies the growth condition

$$\beta\|x - x^*\|^\gamma \leq f(x) - f(x^*), \qquad \forall\ x \text{ such that } f(x) \leq f(x_0),$$

for some scalars $\beta > 0$ and $\gamma > 2$, then for the method of steepest descent with the Armijo rule and other related rules we have

$$f(x_k) - f(x^*) = O\left(\frac{1}{k^{\frac{\gamma}{\gamma-2}}}\right). \tag{2.10}$$

Thus for example, with a quartic order of growth of f ($\gamma = 4$), an $O(1/k^2)$ estimate is obtained for the cost function error after k iterations. The paper [Dun81] provides a more comprehensive analysis of the convergence rate of gradient-type methods based on order of growth conditions, including cases where the convergence rate is linear and faster than linear.

Scaling

To improve the convergence rate of the steepest descent method one may "scale" the gradient $\nabla f(x_k)$ by multiplication with a positive definite symmetric matrix D_k, i.e., use a direction $d_k = -D_k \nabla f(x_k)$, leading to the algorithm

$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k); \tag{2.11}$$

cf. Fig. 2.1.1. Since for $\nabla f(x_k) \neq 0$ we have

$$f'(x_k; d_k) = -\nabla f(x_k)' D_k \nabla f(x_k) < 0,$$

it follows that we still have a cost descent method, as long as the positive stepsize α_k is sufficiently small so that $f(x_{k+1}) < f(x_k)$.

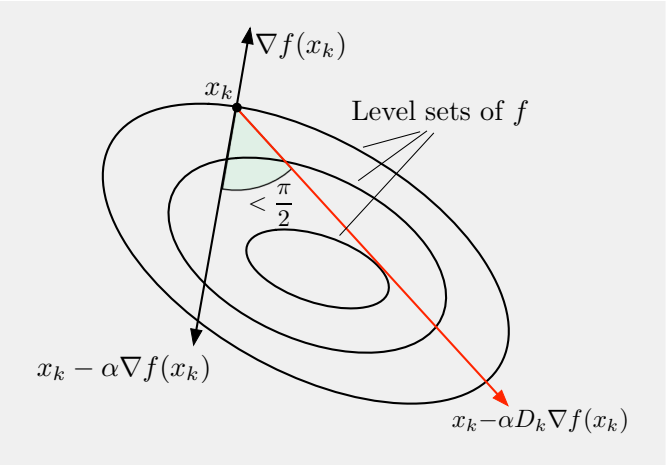

Figure 2.1.1. Illustration of descent directions. Any direction of the form

$$d_k = -D_k \nabla f(x_k),$$

where D_k is a positive definite matrix, is a descent direction because $d_k' \nabla f(x_k) = -d_k' D_k d_k < 0$. In this case d_k makes an angle less than $\pi/2$ with $-\nabla f(x_k)$.

Scaling is a major concept in the algorithmic theory of nonlinear programming. It is motivated by the idea of modifying the "effective condition number" of the problem through a linear change of variables of the form $x = D_k^{1/2} y$. In particular, the iteration (2.11) may be viewed as a steepest descent iteration

$$y_{k+1} = y_k - \alpha \nabla h_k(y_k)$$

for the equivalent problem of minimizing the function $h_k(y) = f\big(D_k^{1/2} y\big)$. For a quadratic problem, where $f(x) = \frac{1}{2} x' Q x - b' x$, the condition number of h_k is the ratio of largest to smallest eigenvalue of the matrix $D_k^{1/2} Q D_k^{1/2}$ (rather than Q).

Much of unconstrained nonlinear programming methodology deals with ways to compute "good" scaling matrices D_k, i.e., matrices that result in fast convergence rate. The "best" scaling in this sense is attained with

$$D_k = \big(\nabla^2 f(x_k)\big)^{-1},$$

assuming that the inverse above exists and is positive definite, which asymptotically leads to an "effective condition number" of 1. This is Newton's method, which will be discussed shortly. A simpler alternative is to use a diagonal approximation to the Hessian matrix $\nabla^2 f(x_k)$, i.e., the diagonal

matrix D_k that has the inverse second partial derivatives

$$\left(\frac{\partial^2 f(x_k)}{(\partial x^i)^2}\right)^{-1}, \qquad i = 1, \ldots, n,$$

along the diagonal. This often improves the performance of the classical gradient method dramatically, by providing automatic scaling of the units in which the components x^i of x are measured, and also facilitates the choice of stepsize – good values of α_k are often chose to 1 (see the subsequent discussion of Newton's method and sources such as [Ber99], Section 1.3).

The nonlinear programming methodology also prominently includes quasi-Newton methods, which construct scaling matrices iteratively, using gradient information collected during the algorithmic process (see nonlinear programming textbooks such as [Pol71], [GMW81], [Lue84], [DeS96], [Ber99], [Fle00], [NoW06], [LuY08]). Some of these methods approximate the full inverse Hessian of f, and eventually attain the fast convergence rate of Newton's method. Other methods use a limited number of gradient vectors from previous iterations (have "limited memory") to construct a relatively crude but still effective approximation to the Hessian of f, and attain a convergence rate that is considerably faster than the one of the unscaled gradient method; see [Noc80], [NoW06].

Gradient Methods with Extrapolation

A variant of the gradient method, known as *gradient method with momentum*, involves extrapolation along the direction of the difference of the preceding two iterates:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}), \tag{2.12}$$

where β_k is a scalar in $[0, 1)$, and we define $x_{-1} = x_0$. When α_k and β_k are chosen to be constant scalars α and β, respectively, the method is known as the *heavy ball method* [Pol64]; see Fig. 2.1.2. This is a sound method with guaranteed convergence under a Lipschitz continuity assumption on ∇f. It can be shown to have faster convergence rate than the corresponding gradient method where α_k is constant and $\beta_k \equiv 0$ (see [Pol87], Section 3.2.1, or [Ber99], Section 1.3). In particular, for a positive definite quadratic problem, and with optimal choices of the constants α and β, the convergence rate of the heavy ball method is linear, and is governed by the formula (2.8) but with $\sqrt{M/m}$ in place of M/m. This is a substantial improvement over the steepest descent method, although the method can still be very slow. Simple examples also suggest that with a momentum term, the steepest descent method is less prone to getting trapped at "shallow" local minima, and deals better with cost functions that are alternately very flat and very steep along the path of the algorithm.

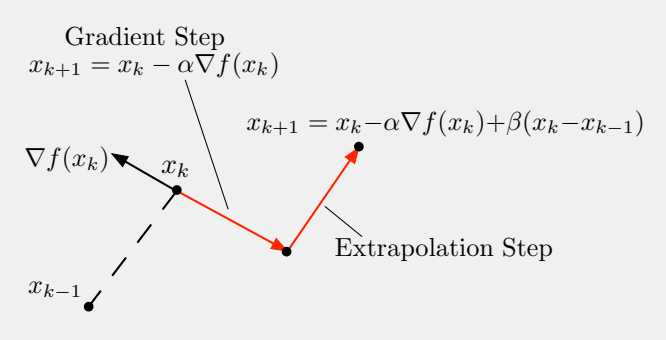

Figure 2.1.2. Illustration of the heavy ball method (2.12), where $\alpha_k \equiv \alpha$ and $\beta_k \equiv \beta$.

A method with similar structure as (2.12), proposed in [Nes83], has received a lot of attention because it has optimal iteration complexity properties under certain conditions, including Lipschitz continuity of ∇f. As we will see in Section 6.2, it improves on the $O(1/k)$ error estimate (2.9) of the gradient method by a factor of $1/k$. The iteration of this method, when applied to unconstrained minimization of a differentiable function f is commonly described in two steps: first an extrapolation step, to compute

$$y_k = x_k + \beta_k(x_k - x_{k-1})$$

with β_k chosen in a special way so that $\beta_k \to 1$, and then a gradient step with constant stepsize α, and gradient calculated at y_k,

$$x_{k+1} = y_k - \alpha\nabla f(y_k).$$

Compared to the method (2.12), it reverses the order of gradient calculation and extrapolation, and uses $\nabla f(y_k)$ in place of $\nabla f(x_k)$.

Conjugate Gradient Methods

There is an interesting connection between the extrapolation method (2.12) and the *conjugate gradient method* for unconstrained differentiable optimization. This is a classical method, with an extensive theory, and the distinctive property that it minimizes an n-dimensional convex quadratic cost function in at most n iterations, each involving a single line minimization. Fast progress is often obtained in much less than n iterations, depending on the eigenvalue structure of the quadratic cost [see e.g., [Ber82a] (Section 1.3.4), or [Lue84] (Chapter 8)]. The method can be implemented in several different ways, for which we refer to textbooks such as [Lue84], [Ber99]. It is a member of the more general class of *conjugate direction methods*, which involve a sequence of exact line searches along directions that are orthogonal with respect to some generalized inner product.

It turns out that if the parameters α_k and β_k in iteration (2.12) are chosen optimally *for each* k so that

$$(\alpha_k, \beta_k) \in \arg \min_{\alpha \in \Re,\, \beta \in \Re} f\big(x_k - \alpha \nabla f(x_k) + \beta(x_k - x_{k-1})\big), \qquad k = 0, 1, \ldots, \tag{2.13}$$

with $x_{-1} = x_0$, the resulting method is an implementation of the conjugate gradient method (see e.g., [Ber99], Section 1.6). By this we mean that if f is a convex quadratic function, *the method (2.12) with the stepsize choice (2.13) generates exactly the same iterates as the conjugate gradient method*, and hence minimizes f in at most n iterations. Finding the optimal parameters according to Eq. (2.13) requires solution of a two-dimensional optimization problem in α and β, which may be impractical in the absence of special structure. However, this optimization is facilitated in some important special cases, which also favor the use of other types of conjugate direction methods.†

There are several other ways to implement the conjugate gradient method, all of which generate identical iterates for quadratic cost functions, but may differ substantially in their behavior for nonquadratic ones. One of them, which resembles the preceding extrapolation methods, is the *method of parallel tangents* or PARTAN, first proposed in the paper [SBK64]. In particular, each iteration of PARTAN involves extrapolation and *two one-dimensional line minimizations*. At the typical iteration, given x_k, we obtain x_{k+1} as follows:

(1) We find a vector y_k that minimizes f over the line

$$\big\{y = x_k - \gamma \nabla f(x_k) \mid \gamma \geq 0\big\}.$$

(2) We generate x_{k+1} by minimizing f over the line that passes through x_{k-1} and y_k.

† Examples of favorably structured problems for conjugate direction methods include cost functions of the form $f(x) = h(Ax)$, where A is a matrix such that the calculation of the vector $y = Ax$ for a given x is far more expensive than the calculation of $h(y)$ and its gradient and Hessian (assuming it exists). Several of the applications described in Sections 1.3 and 1.4 are of this type; see also the papers [NaZ05] and [GoS10], where the application of the subspace minimization method (2.13) and PARTAN are discussed. For such problems, calculation of a stepsize by line minimization along a direction d, as in various types of conjugate direction methods, is relatively inexpensive. In particular, calculation of values, first, and second derivatives of the function $g(\alpha) \equiv f(x + \alpha d) = h(Ax + \alpha Ad)$ requires just two expensive operations: the one-time calculation of the matrix-vector products Ax and Ad. Similarly, minimization over a subspace that passes through x and is spanned by m directions $d_1, \ldots, d_m$, requires the one-time calculation of the matrix-vector products Ax and $Ad_1, \ldots, Ad_m$.

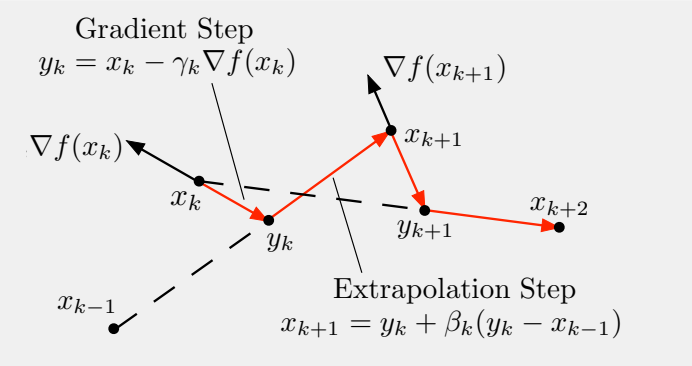

Figure 2.1.3. Illustration of the two-step method

$$y_k = x_k - \gamma_k \nabla f(x_k), \qquad x_{k+1} = y_k + \beta_k(y_k - x_{k-1}).$$

By writing the method equivalently as

$$x_{k+1} = x_k - \gamma_k(1+\beta_k)\nabla f(x_k) + \beta_k(x_k - x_{k-1}),$$

we see that the heavy ball method (2.12) with constant parameters α and β is obtained when $\gamma_k \equiv \alpha/(1+\beta)$ and $\beta_k \equiv \beta$. The PARTAN method is obtained when γ_k and β_k are chosen by line minimization, in which case the corresponding parameter α_k of iteration (2.12) is $\alpha_k = \gamma_k(1+\beta_k)$.

This iteration is a special case of the gradient method with momentum (2.12), corresponding to special choices of α_k and β_k. To see this, observe that we can write iteration (2.12) as a two-step method:

$$y_k = x_k - \gamma_k \nabla f(x_k), \qquad x_{k+1} = y_k + \beta_k(y_k - x_{k-1}),$$

where

$$\gamma_k = \frac{\alpha_k}{1+\beta_k}.$$

Thus starting from x_k, the parameter β_k is determined by the second line search of PARTAN as the optimal stepsize along the line that passes through x_{k-1} and y_k, and then α_k is determined as $\gamma_k(1+\beta_k)$, where γ_k is the optimal stepsize along the line

$$\{x_k - \gamma \nabla f(x_k) \mid \gamma \geq 0\}$$

(cf. Fig. 2.1.3).

The salient property of PARTAN is that when f is convex quadratic it is mathematically equivalent to the conjugate gradient method (it generates exactly the same iterates and terminates in at most n iterations). For this it is essential that the line minimizations are exact, which may

be difficult to guarantee in practice. However, PARTAN seems to be quite resilient to line minimization errors relative to other conjugate gradient implementations. Note that PARTAN ensures at least as large cost reduction at each iteration, as the steepest descent method, since the latter method omits the second line minimization. Thus even for nonquadratic cost functions it tends to perform faster than steepest descent, and often considerably so. We refer to [Lue84], [Pol87], and [Ber99], Section 1.6, for further discussion. These books also address additional issues for the conjugate gradient and other conjugate direction methods, such as alternative implementations, scaling (also called preconditioning), one-dimensional line search algorithms, and rate of convergence.

Newton's Method

In Newton's method the descent direction is

$$d_k = -\big(\nabla^2 f(x_k)\big)^{-1} \nabla f(x_k),$$

provided $\nabla^2 f(x_k)$ exists and is positive definite, so the iteration takes the form

$$x_{k+1} = x_k - \alpha_k \big(\nabla^2 f(x_k)\big)^{-1} \nabla f(x_k).$$

If $\nabla^2 f(x_k)$ is not positive definite, some modification is necessary. There are several possible modifications of this type, for which the reader may consult nonlinear programming textbooks. The simplest one is to add to $\nabla^2 f(x_k)$ a small positive multiple of the identity. Generally, when f is convex, $\nabla^2 f(x_k)$ is positive semidefinite (Prop. 1.1.10 in Appendix B), and this facilitates the implementation of reliable Newton-type algorithms.

The idea in Newton's method is to minimize at each iteration the quadratic approximation of f around the current point x_k given by

$$\tilde{f}_k(x) = f(x_k) + \nabla f(x_k)'(x - x_k) + \tfrac{1}{2}(x - x_k)'\nabla^2 f(x_k)(x - x_k).$$

By setting the gradient of $\tilde{f}_k(x)$ to zero,

$$\nabla f(x_k) + \nabla^2 f(x_k)(x - x_k) = 0,$$

and solving for x, we obtain as next iterate the minimizing point

$$x_{k+1} = x_k - \big(\nabla^2 f(x_k)\big)^{-1} \nabla f(x_k). \tag{2.14}$$

This is the Newton iteration corresponding to a stepsize $\alpha_k = 1$. It follows that, assuming $\alpha_k = 1$, *Newton's method finds the global minimum of a positive definite quadratic function in a single iteration.*

Newton's method typically converges very fast asymptotically, assuming that it converges to a vector x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$

is positive definite, and that a stepsize $\alpha_k = 1$ is used, at least after some iteration. For a simple argument, we may use Taylor's theorem to write

$$0 = \nabla f(x^*) = \nabla f(x_k) + \nabla^2 f(x_k)'(x^* - x_k) + o\big(\|x_k - x^*\|\big).$$

By multiplying this relation with $\big(\nabla^2 f(x_k)\big)^{-1}$ we have

$$x_k - x^* - \big(\nabla^2 f(x_k)\big)^{-1}\nabla f(x_k) = o\big(\|x_k - x^*\|\big),$$

so for the Newton iteration with stepsize $\alpha_k = 1$ we obtain

$$x_{k+1} - x^* = o\big(\|x_k - x^*\|\big),$$

or, for $x_k \neq x^*$,

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = \lim_{k\to\infty} \frac{o\big(\|x_k - x^*\|\big)}{\|x_k - x^*\|} = 0,$$

implying convergence that is faster than linear (also called *superlinear*). This argument can also be used to show *local convergence* to x^* with $\alpha_k \equiv 1$, that is, convergence assuming that x_0 is sufficiently close to x^*.

In implementations of Newton's method, some stepsize rule is often used to ensure cost reduction, but the rule is typically designed so that near convergence we have $\alpha_k = 1$, to ensure that a superlinear convergence rate is attained [assuming $\nabla^2 f(x^*)$ is positive definite at the limit x^*]. Methods that approximate Newton's method also use a stepsize close to 1, and modify the stepsize based on the results of the computation (see sources on nonlinear programming, such as [Ber99], Section 1.4).

The price for the fast convergence of Newton's method is the overhead required to calculate the Hessian matrix, and to solve the linear system of equations

$$\nabla^2 f(x_k) d_k = -\nabla f(x_k)$$

in order to find the Newton direction. There are many iterative algorithms that are patterned after Newton's method, and aim to strike a balance between fast convergence and high overhead (e.g., quasi-Newton, conjugate direction, and others, extensive discussions of which may be found in nonlinear programming textbooks such as [GMW81], [DeS96], [Ber99], [Fle00], [BSS06], [NoW06], [LuY08]).

We finally note that for some problems the special structure of the Hessian matrix can be exploited to facilitate the implementation of Newton's method. For example the Hessian matrix of the dual function of the separable convex programming problem of Section 1.1, when it exists, has particularly favorable structure; see [Ber99], Section 6.1. The same is true for optimal control problems that involve a discrete-time dynamic system and a cost function that is additive over time; see [Ber99], Section 1.9.

Stepsize Rules

There are several methods to choose the stepsize α_k in the scaled gradient iteration (2.11). For example, α_k may be chosen by *line minimization*:

$$\alpha_k \in \arg\min_{\alpha \geq 0} f\big(x_k - \alpha D_k \nabla f(x_k)\big).$$

This can typically be implemented only approximately, with some iterative one-dimensional optimization algorithm; there are several such algorithms (see nonlinear programming textbooks such as [GMW81], [Ber99], [BSS06], [NoW06], [LuY08]).

Our analysis in subsequent chapters of this book will mostly focus on two cases: when α_k is chosen to be *constant*,

$$\alpha_k = \alpha, \qquad k = 0, 1, \ldots,$$

and when α_k is chosen to be *diminishing* to 0, while satisfying the conditions†

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty. \tag{2.15}$$

A convergence analysis for these two stepsize rules is given in the end-of-chapter exercises, and also in Chapter 3, in the context of subgradient methods, as well as in Section 6.1.

We emphasize the constant and diminishing stepsize rules because they are the ones that most readily generalize to nondifferentiable cost functions. However, other stepsize rules, briefly discussed in this chapter, are also important, particularly for differentiable problems, and are used widely. One possibility is the line minimization rule discussed earlier. There are also other rules, which are simple and are based on successive reduction of α_k, until a form of descent is achieved that guarantees convergence. One of these, the *Armijo rule* (first proposed in [Arm66], and sometimes called *backtracking rule*), is popular in unconstrained minimization algorithm implementations. It is given by

$$\alpha_k = \beta^{m_k} s_k,$$

where m_k is the first nonnegative integer m for which

$$f(x_k) - f\big(x_k - \beta^m s_k D_k \nabla f(x_k)\big) \geq \sigma \beta^m s_k \nabla f(x_k)' D_k \nabla f(x_k),$$

† The condition $\sum_{k=0}^{\infty} \alpha_k = \infty$ is needed so that the method can approach the minimum from arbitrarily far, and the condition $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$ is needed so that $\alpha_k \to 0$ and also for technical reasons relating to the convergence analysis (see Section 3.2). If f is a positive definite quadratic, the steepest descent method with a diminishing stepsize α_k satisfying $\sum_{k=0}^{\infty} \alpha_k = \infty$ can be shown to converge to the optimal solution, but at a rate that is slower than linear.

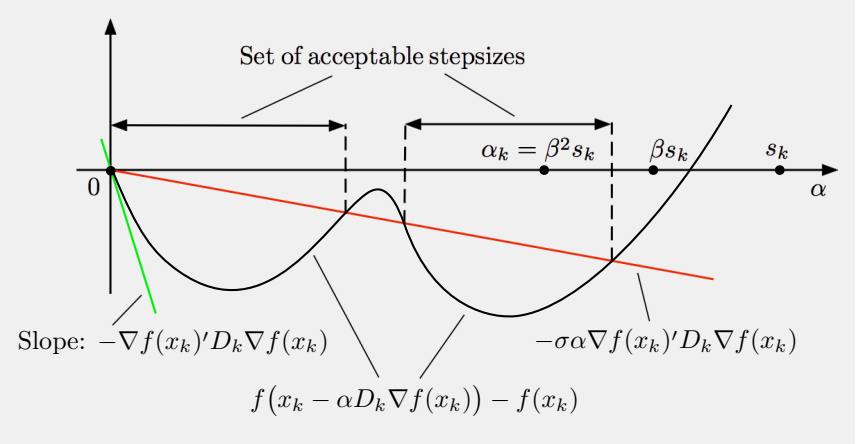

Figure 2.1.4. Illustration of the successive points tested by the Armijo rule along the descent direction $d_k = -D_k \nabla f(x_k)$. In this figure, α_k is obtained as $\beta^2 s_k$ after two unsuccessful trials. Because $\sigma \in (0,1)$, the set of acceptable stepsizes begins with a nontrivial interval interval when $d_k \neq 0$. This implies that if $d_k = 0$, the Armijo rule will find an acceptable stepsize with a finite number of stepsize reductions.

where $\beta \in (0,1)$ and $\sigma \in (0,1)$ are some constants, and $s_k > 0$ is positive initial stepsize, chosen to be either constant or through some simplified search or polynomial interpolation. In other words, starting with an initial trial s_k, the stepsizes $\beta^m s_k$, $m = 0, 1, \ldots$, are tried successively until the above inequality is satisfied for $m = m_k$; see Fig. 2.1.4. We will explore the convergence properties of this rule in the exercises.

Aside from guaranteeing cost function descent, successive reduction rules have the additional benefit of adapting the size of the stepsize α_k to the search direction $-D_k \nabla f(x_k)$, particularly when the initial stepsize s_k is chosen by some simplified search process. We refer to nonlinear programming sources for detailed discussions.

Note that the diminishing stepsize rule does not guarantee cost function descent at each iteration, although it reduces the cost function value once the stepsize becomes sufficiently small. There are also some other rules, often called *nonmonotonic*, which do not explicitly try to enforce cost function descent and have achieved some success, but are based on ideas that we will not discuss in this book; see [GLL86], [BaB88], [Ray93], [Ray97], [BMR00], [DHS06]. An alternative approach to enforce descent without explicitly using stepsizes is based on the trust region methodology for which we refer to book sources such as [Ber99], [CGT00], [Fle00], [NoW06].

2.1.2 Constrained Problems – Feasible Direction Methods

Let us now consider minimizing a differentiable cost function f over a closed convex subset X of $\Re^n$. In a natural form of the cost function descent approach, we may consider generating a feasible sequence $\{x_k\} \subset X$ with an iteration of the form

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2.16}$$

while enforcing cost improvement. However, this is now more complicated because it is not enough for d_k to be a descent direction at x_k. It must also be a *feasible direction* in the sense that $x_k + \alpha d_k$ must belong to X for small enough $\alpha > 0$, in order for the new iterate x_{k+1} to belong to X with suitably small choice of α_k. By multiplying d_k with a positive constant if necessary, this essentially restricts d_k to be of the form $\bar{x}_k - x_k$ for some $\bar{x}_k \in X$ with $\bar{x}_k \neq x_k$. Thus, if f is differentiable, for a feasible descent direction, it is sufficient that

$$d_k = \bar{x}_k - x_k, \qquad \text{for some } \bar{x}_k \in X \text{ with } \nabla f(x_k)'(\bar{x}_k - x_k) < 0.$$

Methods of the form (2.16), where d_k is a feasible descent direction were introduced in the 60s (see e.g., the books [Zou60], [Zan69], [Pol71], [Zou76]), and have been used extensively in applications. We refer to them as *feasible direction methods*, and we give examples of some of the most popular ones.

Conditional Gradient Method

The simplest feasible direction method is to find at iteration k,

$$\bar{x}_k \in \arg\min_{x \in X} \nabla f(x_k)'(x - x_k), \tag{2.17}$$

and set

$$d_k = \bar{x}_k - x_k$$

in Eq. (2.16); see Fig. 2.1.5. Clearly $\nabla f(x_k)'(\bar{x}_k - x_k) \leq 0$, with equality holding only if $\nabla f(x_k)'(x - x_k) \geq 0$ for all $x \in X$, which is a necessary condition for optimality of x_k.

This is the *conditional gradient method* (also known as the *Frank-Wolfe algorithm*) proposed in [FrW56] for convex programming problems with linear constraints, and for more general problems in [LeP65]. The method has been used widely in many contexts, as it is theoretically sound, quite simple, and often convenient. In particular, when X is a polyhedral set, computation of $\bar{x}_k$ requires the solution of a linear program. In some important cases, this linear program has special structure, which results in great simplifications, e.g., in the multicommodity flow problem of Example

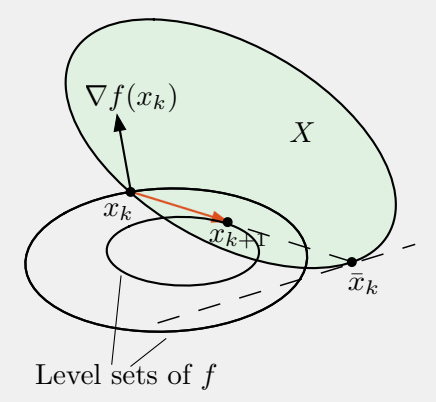

Figure 2.1.5. Illustration of the conditional gradient iteration at x_k. We find $\bar{x}_k$, a point of X that lies farthest along the negative gradient direction $-\nabla f(x_k)$. We then set

$$x_{k+1} = x_k + \alpha_k(\bar{x}_k - x_k),$$

where α_k is a stepsize from $(0, 1]$ (the figure illustrates the case where α_k is chosen by line minimization).

1.4.5 (see the book [BeG92], or the surveys [FlH95], [Pat01]). There has been intensified interest in the conditional gradient method, thanks to applications in machine learning; see e.g., [Cla10], [Jag13], [LuT13], [RSW13], [FrG14], [HJN14], and the references quoted there.

However, the conditional gradient method often tends to converge very slowly relative to its competitors (its asymptotic convergence rate can be slower than linear even for positive definite quadratic programming problems); see [CaC68], [Dun79], [Dun80]. For this reason, other methods with better practical convergence rate properties are often preferred.

One of these methods, is the *simplicial decomposition algorithm* (first proposed independently in [CaG74] and [Hol74]), which will be discussed in detail in Chapter 4. This method is not a feasible direction method of the form (2.16), but instead it is based on multidimensional optimizations over approximations of the constraint set by convex hulls of finite numbers of points. When X is a polyhedral set, it converges in a finite number of iterations, and while this number can potentially be very large, the method often attains practical convergence in very few iterations. Generally, simplicial decomposition can provide an attractive alternative to the conditional gradient method because it tends to be well-suited for the same type of problems [it also requires solution of linear cost subproblems of the form (2.17); see the discussion of Section 4.2].

Somewhat peculiarly, the practical performance of the conditional gradient method tends to improve in highly constrained problems. An explanation for this is given in the papers [Dun79], [DuS83], where it is shown among others that the convergence rate of the method is linear when the cost function is positive definite quadratic, and the constraint set is not polyhedral but rather has a "positive curvature" property (for example it is a sphere). When there are many linear constraints, the constraint set tends to have very many closely spaced extreme points, and has this "positive curvature" property in an approximate sense.

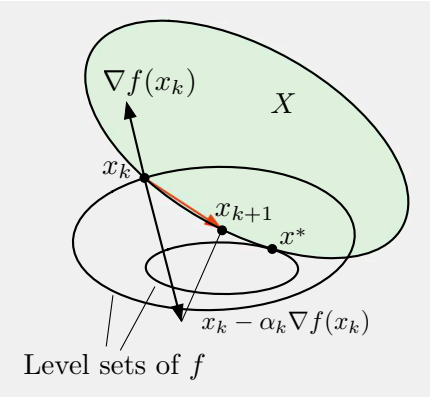

Figure 2.1.6. Illustration of the gradient projection iteration at x_k. We move from x_k along the direction $-\nabla f(x_k)$ and project $x_k - \alpha_k \nabla f(x_k)$ onto X to obtain x_{k+1}. We have

$$\nabla f(x_k)'(x_{k+1} - x_k) \leq 0,$$

and unless $x_{k+1} = x_k$, in which case x_k minimizes f over X, the angle between $\nabla f(x_k)$ and $(x_{k+1} - x_k)$ is strictly greater than 90 degrees, and we have

$$\nabla f(x_k)'(x_{k+1} - x_k) < 0.$$

Gradient Projection Method

Another major feasible direction method, which generally achieves a faster convergence rate than the conditional gradient method, is the *gradient projection method* (originally proposed in [Gol64], [LeP65]), which has the form

$$x_{k+1} = P_X\big(x_k - \alpha_k \nabla f(x_k)\big), \tag{2.18}$$

where $\alpha_k > 0$ is a stepsize and $P_X(\cdot)$ denotes projection on X (the projection is well defined since X is closed and convex; see Fig. 2.1.6).

To get a sense of the validity of the method, note that from the Projection Theorem (Prop. 1.1.9 in Appendix B), we have

$$\nabla f(x_k)'(x_{k+1} - x_k) \leq 0,$$

and by the optimality condition for convex functions (cf. Prop. 1.1.8 in Appendix B), the inequality is strict unless x_k is optimal. Thus $x_{k+1} - x_k$ defines a feasible descent direction at x_k, and based on this fact, we can show the descent property $f(x_{k+1}) < f(x_k)$ when α_k is sufficiently small.

The stepsize α_k is chosen similar to the unconstrained gradient method, i.e., constant, diminishing, or through some kind of reduction rule to ensure cost function descent and guarantee convergence to the optimum; see the convergence analysis of Section 6.1, and [Ber99], Section 2.3, for a detailed discussion and references. Moreover the convergence rate estimates given earlier for unconstrained steepest descent in the positive definite quadratic cost case [cf. Eq. (2.8)] and in the singular case [cf. Eqs. (2.9) and (2.10)] generalize to the gradient projection method under various stepsize rules (see Exercise 2.1 for the former case and [Dun81] for the latter case).

Two-Metric Projection Methods

Despite its simplicity, the gradient projection method has some significant drawbacks:

(a) Its rate of convergence is similar to the one of steepest descent, and is often slow. It is possible to overcome this potential drawback by a form of scaling. This can be accomplished with an iteration of the form

$$x_{k+1} \in \arg\min_{x \in X} \left\{ \nabla f(x_k)'(x - x_k) + \frac{1}{2\alpha_k}(x - x_k)' H_k (x - x_k) \right\}, \tag{2.19}$$

where H_k is a positive definite symmetric matrix and α_k is a positive stepsize. When H_k is the identity, it can be seen that this iteration gives the same iterate x_{k+1} as the unscaled gradient projection iteration (2.18). When $H_k = \nabla^2 f(x_k)$ and $\alpha_k = 1$, we obtain a constrained form of Newton's method (see nonlinear programming sources for analysis; e.g., [Ber99]).

(b) Depending on the nature of X, the projection operation may involve substantial overhead. The projection is simple when H_k is the identity (or more generally, is diagonal), and X consists of simple lower and/or upper bounds on the components of x:

$$X = \big\{ (x^1, \ldots, x^n) \mid \underline{b}^i \leq x^i \leq \bar{b}^i,\ i = 1, \ldots, n \big\}. \tag{2.20}$$

This is an important special case where the use of gradient projection is convenient. Then the projection decomposes to n scalar projections, one for each $i = 1, \ldots, n$: the ith component of x_{k+1} is obtained by projection of the ith component of $x_k - \alpha_k \nabla f(x_k)$,

$$\big(x_k - \alpha_k \nabla f(x_k)\big)^i,$$

onto the interval of corresponding bounds $[\underline{b}^i, \bar{b}^i]$, and is very simple. However, for general nondiagonal scaling the overhead for solving the quadratic programming problem (2.19) is substantial even if X has a simple bound structure of Eq. (2.20).

To overcome the difficulty with the projection overhead, a scaled projection method known as *two-metric projection method* has been proposed for the case of the bound constraints (2.20) in [Ber82a], [Ber82b]. It has a similar form to the scaled gradient method (2.11), and it is given by

$$x_{k+1} = P_X\big(x_k - \alpha_k D_k \nabla f(x_k)\big). \tag{2.21}$$

It is thus a natural and simple adaptation of unconstrained Newton-like methods to bound-constrained optimization, including quasi-Newton methods. The main difficulty here is that an arbitrary positive definite matrix

D_k will not necessarily yield a descent direction. However, it turns out that if some of the off-diagonal terms of D_k that correspond to components of x_k that are at their boundary are set to zero, one can obtain descent (see Exercise 2.8). Furthermore, one can select D_k as the inverse of a partially diagonalized version of the Hessian matrix $\nabla^2 f(x_k)$ and attain the fast convergence rate of Newton's method (see [Ber82a], [Ber82b], [GaB84]).

The idea of simple two-metric projection with partial diagonalization may be generalized to more complex constraint sets, and it has been adapted in [Ber82b], and subsequent papers such as [GaB84], [Dun91], [LuT93b], to problems of the form

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad \underline{b} \leq x \leq \bar{b}, \quad Ax = c, \end{aligned}$$

where A is an $m \times n$ matrix, and $\underline{b}, \bar{b} \in \Re^n$ and $c \in \Re^m$ are given vectors. For example the algorithm (2.21) can be easily modified when the constraint set involves bounds on the components of x together with a few linear constraints, e.g., problems involving a simplex constraint such as

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad 0 \leq x, \quad a'x = c, \end{aligned}$$

where $a \in \Re^n$ and $c \in \Re$, or a Cartesian product of simplexes. For an example of a Newton algorithm of this type, applied to the multicommodity flow problem of Example 1.4.5, see [BeG83]. For representative applications in related large-scale contexts we refer to the papers [Dun91], [LuT93b], [FJS98], [Pyt98], [GeM05], [OJW05], [TaP13], [WSK14].

The advantage that the two-metric projection approach can offer is to identify quickly the constraints that are active at an optimal solution. After this happens, the method reduces essentially to an unconstrained scaled gradient method (possibly Newton method, if D_k is a partially diagonalized Hessian matrix), and attains a fast convergence rate. This property has also motivated variants of the two-metric projection method for problems involving ℓ_1-regularization, such as the ones of Example 1.3.2; see [SFR09], [Sch10], [GKX10], [SKS12], [Lan14].

Block Coordinate Descent

The preceding methods require the computation of the gradient and possibly the Hessian of the cost function at each iterate. An alternative descent approach that does not require derivatives or other direction calculations is the classical *block coordinate descent* method, which we will briefly describe here and consider further in Section 6.5. The method applies to the problem

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in X, \end{aligned}$$

where $f : \Re^n \mapsto \Re$ is a differentiable function, and X is a Cartesian product of closed convex sets $X_1, \ldots, X_m$:

$$X = X_1 \times X_2 \times \cdots \times X_m.$$

The vector x is partitioned as

$$x = (x^1, x^2, \ldots, x^m),$$

where each x^i belongs to $\Re^{n_i}$, so the constraint $x \in X$ is equivalent to

$$x^i \in X_i, \qquad i = 1, \ldots, m.$$

The most common case is when $n_i = 1$ for all i, so the components x^i are scalars. The method involves minimization with respect to a single component x^i at each iteration, with all other components kept fixed.

In an example of such a method, given the current iterate $x_k = (x_k^1, \ldots, x_k^m)$, we generate the next iterate $x_{k+1} = (x_{k+1}^1, \ldots, x_{k+1}^m)$, according to the "cyclic" iteration

$$x_{k+1}^i \in \arg\min_{\xi \in X_i} f(x_{k+1}^1, \ldots, x_{k+1}^{i-1}, \xi, x_k^{i+1}, \ldots, x_k^m), \quad i = 1, \ldots, m. \quad (2.22)$$

Thus, at each iteration, the cost is minimized with respect to each of the "block coordinate" vectors x^i, taken one-at-a-time in cyclic order.

Naturally, the method makes practical sense only if it is possible to perform this minimization fairly easily. This is frequently so when each x^i is a scalar, but there are also other cases of interest, where x^i is a multidimensional vector. Moreover, the method can take advantage of special structure of f; an example of such structure is a form of "sparsity," where f is the sum of component functions, and for each i, only a relatively small number of the component functions depend on x^i, thereby simplifying the minimization (2.22). The following is an example of a classical algorithm that can be viewed as a special case of block coordinate descent.

Example 2.1.1 (Parallel Projections Algorithm)

We are given m closed convex sets $X_1, \ldots, X_m$ in $\Re^n$, and we want to find a point in their intersection. This problem can equivalently be written as

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^m \|y^i - x\|^2 \\ &\text{subject to} \quad x \in \Re^n, \ \ y^i \in X_i, \ i = 1, \ldots, m, \end{aligned}$$

where the variables of the optimization are $x, y^1, \ldots, y^m$ (the optimal solutions of this problem are the points in the intersection $\cap_{i=1}^m X_i$, if this intersection is nonempty). A block coordinate descent algorithm iterates on each of the vectors $y^1, \ldots, y^m$ in parallel according to

$$y_{k+1}^i = P_{X_i}(x_k), \qquad i = 1, \ldots, m,$$

and then iterates with respect to x according to

$$x_{k+1} = \frac{y_{k+1}^1 + \cdots + y_{k+1}^m}{m},$$

which minimizes the cost function with respect to x when each y^i is fixed at y_{k+1}^i.

Here is another example where the coordinate descent method takes advantage of decomposable structure.

Example 2.1.2 (Hierarchical Decomposition)

Consider an optimization problem of the form

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^m \big(h_i(y^i) + f_i(x, y^i)\big) \\ &\text{subject to} \quad x \in X, \qquad y^i \in Y_i, \quad i = 1, \ldots, m, \end{aligned}$$

where X and Y_i, $i = 1, \ldots, m$, are closed, convex subsets of corresponding Euclidean spaces, and h_i, f_i are given functions, assumed differentiable. This problem is associated with a paradigm of optimization of a system consisting of m subsystems, with a cost function $h_i + f_i$ associated with the operations of the ith subsystem. Here y^i is viewed as a vector of local decision variables that influences the cost of the ith subsystem only, and x is viewed as a vector of global or coordinating decision variables that affects the operation of all the subsystems.

The block coordinate descent method has the form

$$y_{k+1}^i \in \arg\min_{y^i \in Y_i} \big\{h_i(y^i) + f_i(x_k, y^i)\big\}, \qquad i = 1, \ldots, m,$$

$$x_{k+1} \in \arg\min_{x \in X} \sum_{i=1}^m f_i(x, y_{k+1}^i).$$

The method has a natural real-life interpretation: at each iteration, each subsystem optimizes its own cost, viewing the global variables as fixed at their current values, and then the coordinator optimizes the overall cost for the current values of the local variables (without having to know the "local" cost functions h_i of the subsystems).

In the absence of special structure of f, differentiability is essential for the validity of the coordinate descent method; this can be verified with simple examples. In our convergence analysis of Chapter 6, we will also require a form of strict convexity of f along each block component, as first suggested in the book [Zan69] (subtle examples of nonconvergence have been constructed in the absence of a property of this kind [Pow73]). We

note, however, there are some interesting cases where nondifferentiabilities with special structure can be dealt with; see Section 6.5.

There are several variants of the method, which incorporate various descent algorithms in the solution of the block minimizations (2.22). Another type of variant is one where the block components are iterated in an irregular order instead of a fixed cyclic order. In fact there is a substantial theory of asynchronous distributed versions of coordinate descent, for which we refer to the parallel and distributed algorithms book [BeT89a], and the sources quoted there; see also the discussion in Sections 2.1.6 and 6.5.2.

2.1.3 Nondifferentiable Problems – Subgradient Methods

We will now briefly consider the minimization of a convex nondifferentiable cost function $f : \Re^n \mapsto \Re$ (optimization of a nonconvex and nondifferentiable function is a far more complicated subject, which we will not address in this book). It is possible to generalize the steepest descent approach so that when f is nondifferentiable at x_k, we use a direction d_k that minimizes the directional derivative $f'(x_k; d)$ subject to $\|d\| \leq 1$,

$$d_k \in \arg \min_{\|d\| \leq 1} f'(x_k; d).$$

Unfortunately, this minimization (or more generally finding a descent direction) may involve a nontrivial computation. Moreover, there is a worrisome theoretical difficulty: the method may get stuck far from the optimum, depending on the stepsize rule. An example is given in Fig. 2.1.7, where the stepsize is chosen using the minimization rule

$$\alpha_k \in \arg \min_{\alpha \geq 0} f(x_k + \alpha d_k).$$

In this example, *the algorithm fails even though it never encounters a point where f is nondifferentiable,* which suggests that convergence questions in convex optimization are delicate and should not be treated lightly. The problem here is a lack of continuity: the steepest descent direction may undergo a large/discontinuous change close to the convergence limit. By contrast, this would not happen if f were continuously differentiable at the limit, and in fact the steepest descent method with the minimization stepsize rule has sound convergence properties when used for differentiable functions.

Because the implementation of cost function descent has the limitations outlined above, a different kind of descent approach, based on the notion of subgradient, is often used when f is nondifferentiable. The theory of subgradients of extended real-valued functions is outlined in Section 5.4 of Appendix B, as developed in the textbook [Ber09]. The properties of

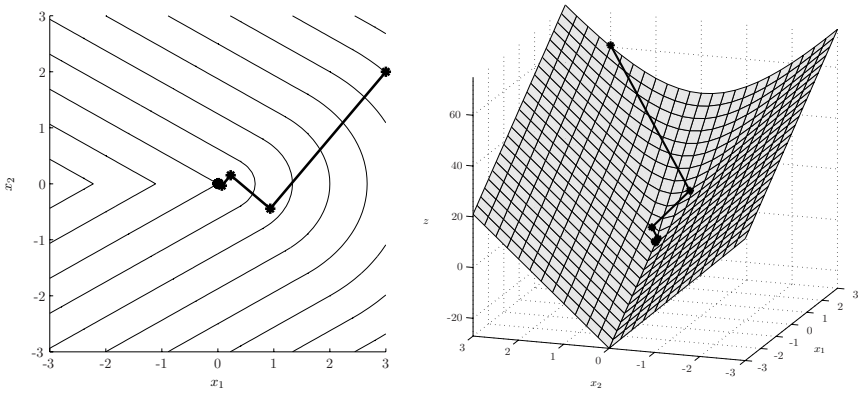

Figure 2.1.7. An example of failure of the steepest descent method with the line minimization stepsize rule for a convex nondifferentiable cost function [Wol75]. Here we have the two-dimensional cost function

$$f(x_1, x_2) = \begin{cases} 5(9x_1^2 + 16x_2^2)^{1/2} & \text{if } x_1 > |x_2|, \\ 9x_1 + 16|x_2| & \text{if } x_1 \leq |x_2|, \end{cases}$$

shown in the figure. Consider the method that moves in the direction of steepest descent from the current point, with the stepsize determined by cost minimization along that direction (this can be done analytically). Suppose that the algorithm starts anywhere within the set

$$\big\{(x_1, x_2) \mid x_1 > |x_2| > (9/16)^2 |x_1|\big\}.$$

The generated iterates are shown in the figure, and it can be verified that they converge to the nonoptimal point $(0, 0)$.

subgradients of real-valued convex functions will also be discussed in detail in Section 3.1.

In the most common subgradient method (first proposed and analyzed in the mid 60s by Shor in a series of papers, and later in the books [Sho85], [Sho98]), an arbitrary subgradient g_k of f at x_k is used in an iteration of the form

$$x_{k+1} = x_k - \alpha_k g_k, \tag{2.23}$$

where α_k is a positive stepsize. The method, together with its many variations, will be discussed extensively in this book, starting with Chapter 3. We will see that while it may not yield a cost reduction for any value of α_k it has another descent property, which enhances the convergence process: at any nonoptimal point x_k, it satisfies

$$\text{dist}(x_{k+1}, X^*) < \text{dist}(x_k, X^*)$$

for a sufficiently small stepsize α_k, where $\text{dist}(x, X^*)$ denotes the Euclidean minimum distance of x from the optimal solution set X^*.

Stepsize Rules and Convergence Rate

There are several methods to choose the stepsize α_k in the subgradient iteration (2.23), some of which will be discussed in more detail in Chapter 3. Among these are:

(a) α_k is chosen to be a positive *constant*,

$$\alpha_k = \alpha, \qquad k = 0, 1, \ldots.$$

In this case only approximate convergence can be guaranteed, i.e., convergence to a neighborhood of the optimum whose size depends on α. Moreover the convergence rate may be slow. However, there is an important case where some favorable results can be shown. This is when a so called *sharp minimum* condition holds, i.e., for some $\beta > 0$,

$$f^* + \beta \min_{x^* \in X^*} \|x - x^*\| \leq f(x), \qquad \forall\, x \in X, \tag{2.24}$$

where f^* is the optimal value (see Exercise 3.10). We will prove in Prop. 5.1.6 that this condition holds when f and X are polyhedral, as for example in dual problems arising in integer programming.

(b) α_k is chosen to be *diminishing* to 0, while satisfying the conditions

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Then exact convergence can be guaranteed, but the convergence rate is sublinear, even for polyhedral problems, and typically very slow.

There are also more sophisticated stepsize rules, which are based on estimation of f^* (see Section 3.2, and [BNO03] for a detailed account). Still, unless the condition (2.24) holds, the convergence rate can be very slow relative to other methods. On the other hand in the presence of special structure, such as in additive cost problems, incremental versions of subgradient methods (see Section 2.1.5) may perform satisfactorily.

2.1.4 Alternative Descent Methods

Aside from methods that are based on gradients or subgradients, like the ones of the preceding sections, there are some other approaches to effect cost function descent. A major approach, which applies to any convex cost function is the *proximal algorithm*, to be discussed in detail in Chapter 5. This algorithm embodies both the cost improvement and the approximation ideas. In its basic form, it approximates the minimization of a closed proper convex function $f : \Re^n \mapsto (-\infty, \infty]$ with another minimization that involves a quadratic term. It is given by

$$x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ f(x) + \frac{1}{2c_k} \|x - x_k\|^2 \right\}, \tag{2.25}$$

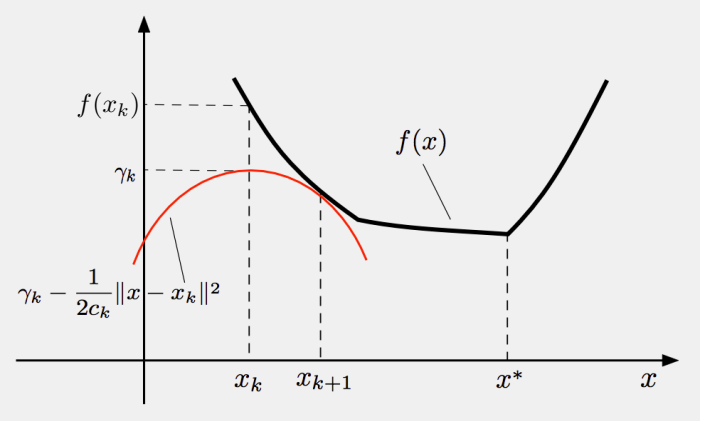

Figure 2.1.8. Illustration of the proximal algorithm (2.25) and its descent property. The minimum of $f(x)+\frac{1}{2c_k}\|x-x_k\|^2$ is attained at the unique point x_{k+1} at which the graph of the quadratic function $-\frac{1}{2c_k}\|x-x_k\|^2$, raised by the amount

$$\gamma_k = f(x_{k+1}) + \frac{1}{2c_k}\|x_{k+1} - x_k\|^2,$$

just touches the graph of f. Since $\gamma_k < f(x_k)$, it follows that $f(x_{k+1}) < f(x_k)$, unless x_k minimizes f, which happens if and only if $x_{k+1} = x_k$.

where x_0 is an arbitrary starting point and c_k is a positive scalar parameter (see Fig. 2.1.8). One of the motivations for the algorithm is that it "regularizes" the minimization of f: the quadratic term in Eq. (2.25) when added to f makes it strictly convex with compact level sets, so it has a unique minimum (cf. Prop. 3.1.1 and Prop. 3.2.1 in Appendix B).

The algorithm has an inherent descent character, which facilitates its combination with other algorithmic schemes. To see this note that since $x = x_{k+1}$ gives a lower value of $f(x) + \frac{1}{2c_k}\|x - x_k\|^2$ than $x = x_k$, we have

$$f(x_{k+1}) + \frac{1}{2c_k}\|x_{k+1} - x_k\|^2 \leq f(x_k).$$

It follows that $\{f(x_k)\}$ is monotonically nonincreasing; see also Fig. 2.1.8.

There are several variations of the proximal algorithm, which will be discussed in Chapters 5 and 6. Some of these variations involve modification of the proximal minimization problem of Eq. (2.25), motivated by the need for a convenient solution of this problem. Here are some examples:

(a) The use of a nonquadratic proximal term $D_k(x; x_k)$ in Eq. (2.25), in place of $(1/2c_k)\|x - x_k\|^2$, i.e., the iteration

$$x_{k+1} \in \arg\min_{x \in \Re^n} \{f(x) + D_k(x; x_k)\}. \tag{2.26}$$

This approach may be useful when D_k has a special form that matches the structure of f.

(b) Linear approximation of f using its gradient at x_k

$$f(x) \approx f(x_k) + \nabla f(x_k)'(x - x_k),$$

assuming that f is differentiable. Then, in place of Eq. (2.26), we obtain the iteration

$$x_{k+1} \in \arg\min_{x \in \Re^n} \big\{ f(x_k) + \nabla f(x_k)'(x - x_k) + D_k(x; x_k) \big\}.$$

When the proximal term $D_k(x; x_k)$ is the quadratic $(1/2c_k)\|x - x_k\|^2$, this iteration can be seen to be equivalent to the gradient projection iteration (2.18):

$$x_{k+1} = P_X\big(x_k - c_k \nabla f(x_k)\big),$$

but there are other choices of D_k that lead to interesting methods, known as *mirror descent algorithms*.

(c) The *proximal gradient algorithm*, which applies to the problem

$$\begin{aligned} &\text{minimize} \quad f(x) + h(x) \\ &\text{subject to } x \in \Re^n, \end{aligned}$$

where $f : \Re^n \mapsto \Re$ is a differentiable convex function, and $h : \Re^n \mapsto (-\infty, \infty]$ is a closed proper convex function. This algorithm combines ideas from the gradient projection method and the proximal method. It replaces f with a linear approximation in the proximal minimization, i.e.,

$$x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ \nabla f(x_k)'(x - x_k) + h(x) + \frac{1}{2\alpha_k}\|x - x_k\|^2 \right\}, \tag{2.27}$$

where $\alpha_k > 0$ is a parameter. Thus when f is a linear function, we obtain the proximal algorithm for minimizing $f + h$. When h is the indicator function of a closed convex set, we obtain the gradient projection method. Note that there is an alternative/equivalent way to write the algorithm (2.27):

$$z_k = x_k - \alpha_k \nabla f(x_k), \quad x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ h(x) + \frac{1}{2\alpha_k}\|x - z_k\|^2 \right\}, \tag{2.28}$$

as can be verified by expanding the quadratic

$$\|x - z_k\|^2 = \big\|x - x_k + \alpha_k \nabla f(x_k)\big\|^2.$$

Thus the method alternates gradient steps on f with proximal steps on h. The advantage that this method may have over the proximal algorithm is that the proximal step in Eq. (2.28) is executed with h rather than with $f + h$, and this may be significant if h has simple/favorable structure (e.g., h is the ℓ_1 norm or a distance function to a simple constraint set), while f has unfavorable structure. Under relatively mild assumptions, it can be shown that the method has a cost function descent property, provided the stepsize α is sufficiently small (see Section 6.3).

In Section 6.7, we will also discuss another descent approach, called *ϵ-descent*, which aims to avoid the difficulties due to the discontinuity of the steepest descent direction (cf. Fig. 2.1.7). This is done by obtaining a descent direction via projection of the origin on an ϵ-subdifferential, an enlarged version of the subdifferential. The method is theoretically interesting and will be used to establish conditions for strong duality in *extended monotropic programming*, an important class of problems with partially separable structure, to be discussed in Section 4.4.

Finally, we note that there are a few types of descent methods that we will not discuss at all, either because they are based on ideas that do not connect well with convexity, or because they are not well suited for the type of large-scale problems that we emphasize in this book. Included are direct search methods that do not use derivatives, such as the Nelder-Mead simplex algorithm [DeT91], [Tse95], [LRW98], [NaT02], feasible direction methods such as reduced gradient and gradient projection methods based on manifold suboptimization [GiM74], [GMW81], [MoT89], and sequential quadratic programming methods [Ber82a], [Ber99], [NoW06]. Some of these methods have extensive literature and applications, but are beyond our scope.

2.1.5 Incremental Algorithms

An interesting form of approximate gradient, or more generally subgradient method, is an *incremental* variant, which applies to minimization over a closed convex set X of an additive cost function of the form

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

where the functions $f_i : \Re^n \mapsto \Re$ are either differentiable or convex and nondifferentiable. We mentioned several contexts where cost functions of this type arise in Section 1.3. The idea of the incremental approach is to sequentially take steps along the subgradients of the component functions f_i, with intermediate adjustment of x after processing each f_i.

Incremental methods are interesting when m is very large, so a full subgradient step is very costly. For such problems one hopes to make

progress with approximate but much cheaper incremental steps. Incremental methods are also well-suited for problems where m is large and the component functions f_i become known sequentially, over time. Then one may be able to operate on each component as it reveals itself, without waiting for the other components to become known, i.e., in *an on-line fashion*.

In a common type of incremental subgradient method, *an iteration is viewed as a cycle of m subiterations*. If x_k is the vector obtained after k cycles, the vector x_{k+1} obtained after one more cycle is

$$x_{k+1} = \psi_{m,k}, \tag{2.29}$$

where starting with

$$\psi_{0,k} = x_k,$$

we obtain $\psi_{m,k}$ after the m steps

$$\psi_{i,k} = P_X(\psi_{i-1,k} - \alpha_k g_{i,k}), \qquad i = 1, \ldots, m, \tag{2.30}$$

with $g_{i,k}$ being a subgradient of f_i at $\psi_{i-1,k}$ [or the gradient $\nabla f_i(\psi_{i-1,k})$ in the differentiable case].

In a *randomized* version of the method, given x_k at iteration k, an index i_k is chosen from the set $\{1, \ldots, m\}$ randomly, and the next iterate x_{k+1} is generated by

$$x_{k+1} = P_X(x_k - \alpha_k g_{i_k}), \qquad i = 1, \ldots, m, \tag{2.31}$$

where g_{i_k} is a subgradient of f_{i_k} at x_k. Here it is important that all indexes are chosen with equal probability. It turns out that there is a rate of convergence advantage for this and other types of randomization, as we will discuss in Section 6.4.2. We will ignore for the moment the possibility of randomizing the component selection, and assume cyclic selection as in Eqs. (2.29)-(2.30).

In the present section we will explain the ideas underlying incremental methods by focusing primarily on the case where the component functions f_i are differentiable. We will thus consider methods that compute at each step a component gradient ∇f_i and possibly Hessian matrix $\nabla^2 f_i$. We will discuss the case where f_i may be nondifferentiable in Section 6.4, after the analysis of nonincremental subgradient methods to be given in Section 3.2.

Incremental Gradient Method

Assume that the component functions f_i are differentiable. We refer to the method

$$x_{k+1} = \psi_{m,k}, \tag{2.32}$$

where starting with $\psi_{0,k} = x_k$, we generate $\psi_{m,k}$ after the m steps

$$\psi_{i,k} = P_X\big(\psi_{i-1,k} - \alpha_k \nabla f_i(\psi_{i-1,k})\big), \qquad i = 1, \ldots, m, \tag{2.33}$$

[cf. (2.29)-(2.30)], as the *incremental gradient method*. A well known and important example of such a method is the following. Together with its many variations, it is widely used in computerized imaging; see e.g., the book [Her09].

Example 2.1.3: (Kaczmarz Method)

Let

$$f_i(x) = \frac{1}{2\|c_i\|^2}(c_i'x - b_i)^2, \qquad i = 1, \ldots, m,$$

where c_i are given nonzero vectors in $\Re^n$ and b_i are given scalars, so we have a linear least squares problem. The constant term $1/(2\|c_i\|^2)$ multiplying each of the squared functions $(c_i'x - b_i)^2$ serves a scaling purpose: with its inclusion, all the components f_i have a Hessian matrix

$$\nabla^2 f_i(x) = \frac{1}{\|c_i\|^2} c_i c_i'$$

with trace equal to 1. This type of scaling is often used in least squares problems (see [Ber99] for explanations). The incremental gradient method (2.32)-(2.33) takes the form $x_{k+1} = \psi_{m,k}$, where $\psi_{m,k}$ is obtained after the m steps

$$\psi_{i,k} = \psi_{i-1,k} - \frac{\alpha_k}{\|c_i\|^2}(c_i'\psi_{i-1,k} - b_i)c_i, \qquad i = 1, \ldots, m, \tag{2.34}$$

starting with $\psi_{0,k} = x_k$ (see Fig. 2.1.9).

The stepsize α_k may be chosen in a number of different ways, but if α_k is chosen identically equal to 1, $\alpha_k \equiv 1$, we obtain the Kaczmarz method, which dates to 1937 [Kac37]; see Fig. 2.1.9(a). The interpretation of the iteration (2.34) in this case is very simple: *$\psi_{i,k}$ is obtained by projecting $\psi_{i,k-1}$ onto the hyperplane defined by the single equation $c_i'x = b_i$*. Indeed from Eq. (2.34) with $\alpha_k = 1$, it is easily verified that $c_i'\psi_{i,k} = b_i$ and that $\psi_{i,k} - \psi_{i,k-1}$ is orthogonal to the hyperplane, since it is proportional to its normal c_i. (There are also other related methods involving alternating projections on subspaces or other convex sets, one of them attributed to von Neumann from 1933; see Section 6.4.4.)

If the system of equations $c_i'x = b_i$, $i = 1, \ldots, m$, is consistent, i.e., has a unique solution x^*, then the unique minimum of $\sum_{i=1}^m f_i(x)$ is x^*. In this case it turns out that for a constant stepsize $\alpha_k \equiv \alpha$, with $0 < \alpha < 2$, the method converges to x^*. The convergence process is illustrated in Fig. 2.1.9(b) for the case $\alpha_k \equiv 1$: the distance $\|\psi_{i,k} - x^*\|$ is guaranteed not to increase for any i within cycle k, and to strictly decrease for at least one i, so x_{k+1} will be closer to x^* than x_k (assuming $x_k \neq x^*$). Generally, the order in which the equations are taken up for iteration can affect significantly the

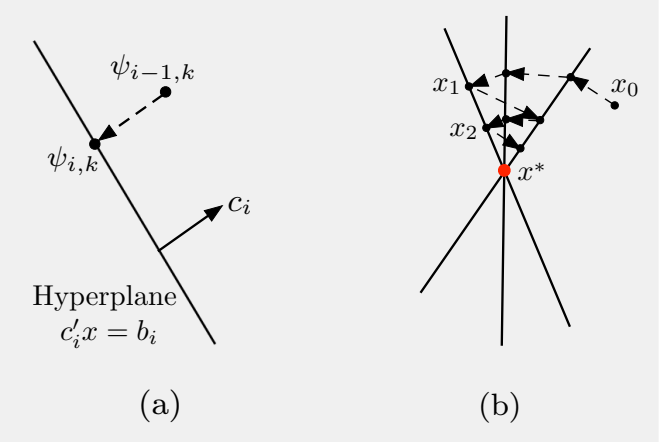

Figure 2.1.9. Illustration of the Kaczmarz method (2.34) with unit stepsize $\alpha_k \equiv 1$: (a) $\psi_{i,k}$ is obtained by projecting $\psi_{i-1,k}$ onto the hyperplane defined by the single equation $c_i'x = b_i$. (b) The convergence process for the case where the system of equations $c_i'x = b_i$, $i = 1, \ldots, m$, is consistent and has a unique solution x^*. Here $m = 3$, and x_k is the vector obtained after k cycles through the equations. Each incremental iteration decreases the distance to x^*, unless the current iterate lies on the hyperplane defined by the corresponding equation.

performance. In particular, faster convergence can be shown if the order is randomized in a special way; see [StV09].

If the system of equations

$$c_i'x = b_i, \qquad i = 1, \ldots, m,$$

is inconsistent, the method does not converge with a constant stepsize; see Fig. 2.1.10. In this case a diminishing stepsize α_k is necessary for convergence to an optimal solution. These convergence properties will be discussed further later in this section, and in Chapters 3 and 6.

Convergence Properties of Incremental Methods

The motivation for the incremental approach is faster convergence. In particular, we hope that far from the solution, a single cycle of the incremental gradient method will be as effective as several (as many as m) iterations of the ordinary gradient method (think of the case where the components f_i are similar in structure). Near a solution, however, the incremental method may not be as effective. Still, the frequent superiority of the incremental method when far from convergence can be a decisive advantage for problems where solution accuracy is not of paramount importance.

To be more specific, we note that there are two complementary performance issues to consider in comparing incremental and nonincremental methods:

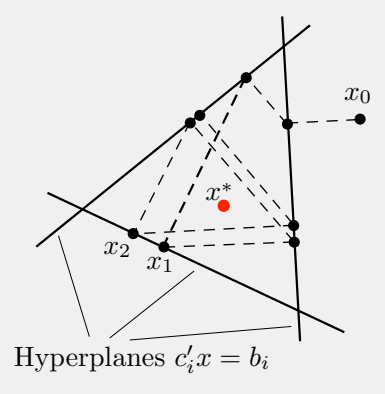

Figure 2.1.10. Illustration of the Kaczmarz method (2.34) with $\alpha_k \equiv 1$ for the case where the system of equations $c_i'x = b_i$, $i = 1, \ldots, m$, is inconsistent. In this figure there are three equations with corresponding hyperplanes as shown. The method approaches a neighborhood of the optimal solution, and then oscillates. A similar behavior would occur if the stepsize α_k were a constant $\alpha \in (0, 1)$, except that the size of the oscillation would diminish with α.

(a) *Progress when far from convergence.* Here the incremental method can be much faster. For an extreme case let $X = \Re^n$ (no constraints), and take m large and all components f_i identical to each other. Then an incremental iteration requires m times less computation than a classical gradient iteration, but gives exactly the same result, when the stepsize is appropriately scaled to be m times larger. While this is an extreme example, it reflects the essential mechanism by which incremental methods can be much superior: far from the minimum a single component gradient will point to "more or less" the right direction, at least most of the time.

(b) *Progress when close to convergence.* Here the incremental method can be inferior. As a case in point, assume that all components f_i are differentiable functions. Then the nonincremental gradient projection method can be shown to converge with a constant stepsize under reasonable assumptions, as we will see in Section 6.1. However, the incremental method requires a diminishing stepsize, and its ultimate rate of convergence can be much slower. When the component functions f_i are nondifferentiable, both the nonincremental and the incremental subgradient methods require a diminishing stepsize. The nonincremental method tends to require a smaller number of iterations, but each of the iterations involves all the components f_i and thus larger computation overhead, so that on balance, in terms of computation time, the incremental method tends to perform better.

As an illustration consider the following example.

Example 2.1.4:

Consider a scalar linear least squares problem where the components f_i have the form

$$f_i(x) = \tfrac{1}{2}(c_i x - b_i)^2, \qquad x \in \Re,$$

where c_i and b_i are given scalars with $c_i \neq 0$ for all i. The minimum of each of the components f_i is

$$x_i^* = \frac{b_i}{c_i},$$

while the minimum of the least squares cost function $f = \sum_{i=1}^m f_i$ is

$$x^* = \frac{\sum_{i=1}^m c_i b_i}{\sum_{i=1}^m c_i^2}.$$

It can be seen that x^* lies within the range of the component minima

$$R = \left[\min_i x_i^*,\ \max_i x_i^*\right],$$

and that for all x *outside* the region R, the gradient

$$\nabla f_i(x) = (c_i x - b_i)c_i$$

has the same sign as $\nabla f(x)$ (see Fig. 2.1.11). As a result, when outside the region R, the incremental gradient method

$$\psi_i = \psi_{i-1} - \alpha_k(c_i\psi_{i-1} - b_i)c_i$$

approaches x^* at each step, provided the stepsize α_k is small enough. In fact it is sufficient that

$$\alpha_k \leq \min_i \frac{1}{c_i^2}.$$

However, *for x inside the region R, the ith step of a cycle of the incremental gradient method need not make progress.* It will approach x^* (for small enough stepsize α_k) only if the current point ψ_{i-1} does not lie in the interval connecting x_i^* and x^*. This induces an oscillatory behavior within R, and as a result, the incremental gradient method will typically not converge to x^* unless $\alpha_k \to 0$.

Let us now compare the incremental gradient method with the nonincremental version, which takes the form

$$x_{k+1} = x_k - \alpha_k \sum_{i=1}^m (c_i x_k - b_i)c_i.$$

It can be shown that this method converges to x^* for any constant stepsize $\alpha_k \equiv \alpha$ satisfying

$$0 < \alpha \leq \frac{1}{\sum_{i=1}^m c_i^2}.$$

On the other hand, for x outside the region R, an iteration of the nonincremental method need not make more progress towards the solution than a single step of the incremental method. In other words, with comparably intelligent stepsize choices, *far from the solution (outside R), a single cycle through the entire set of component functions by the incremental method is roughly as effective as m iterations by the nonincremental method, which require m times as many component gradient calculations.*

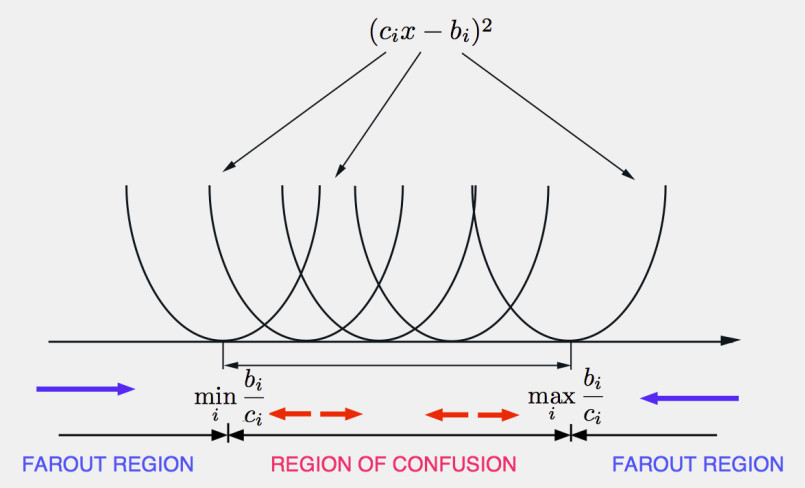

Figure 2.1.11. Illustrating the advantage of incrementalism when far from the optimal solution. The region of component minima

$$R = \left[\min_i x_i^*,\ \max_i x_i^*\right],$$

is labeled as the "region of confusion." It is the region where the method does not have a clear direction towards the optimum. The ith step in an incremental gradient cycle is a gradient step for minimizing $(c_i x - b_i)^2$, so if x lies outside the region of component minima $R = \left[\min_i x_i^*,\ \max_i x_i^*\right]$, (labeled as the "farout region") and the stepsize is small enough, progress towards the solution x^* is made.

Example 2.1.5:

The preceding example assumes that each component function f_i has a minimum, so that the range of component minima is defined. In cases where the components f_i have no minima, a similar phenomenon may occur. As an example consider the case where f is the sum of increasing and decreasing convex exponentials, i.e.,

$$f_i(x) = a_i e^{b_i x}, \qquad x \in \Re,$$

where a_i and b_i are scalars with $a_i > 0$ and $b_i \neq 0$. Let

$$I^+ = \{i \mid b_i > 0\}, \qquad I^- = \{i \mid b_i < 0\},$$

and assume that I^+ and I^- have roughly equal numbers of components. Let also x^* be the minimum of $\sum_{i=1}^m f_i$.

Consider the incremental gradient method that given the current point, call it x_k, chooses some component f_{i_k} and iterates according to the incremental iteration

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k).$$

Then it can be seen that if $x_k >> x^*$, x_{k+1} will be substantially closer to x^* if $i \in I^+$, and negligibly further away than x^* if $i \in I^-$. The net effect, averaged over many incremental iterations, is that if $x_k >> x^*$, an incremental gradient iteration makes roughly one half the progress of a full gradient iteration, with m times less overhead for calculating gradients. The same is true if $x_k << x^*$. On the other hand as x_k gets closer to x^* the advantage of incrementalism is reduced, similar to the preceding example. In fact in order for the incremental method to converge, a diminishing stepsize is necessary, which will ultimately make the convergence slower than the one of the nonincremental gradient method with a constant stepsize.

The preceding examples rely on x being one-dimensional, but in many multidimensional problems the same qualitative behavior can be observed. In particular, the incremental gradient method, by processing the ith component f_i, can make progress towards the solution in the region where the component function gradient $\nabla f_i(\psi_{i-1})$ makes an angle less than 90 degrees with the full cost function gradient $\nabla f(\psi_{i-1})$. If the components f_i are not "too dissimilar," this is likely to happen in a region of points that are not too close to the optimal solution set.

Stepsize Selection

The choice of the stepsize α_k plays an important role in the performance of incremental gradient methods. On close examination, it turns out that the iterate differential $x_k - x_{k+1}$ corresponding to a full cycle of the incremental gradient method, and the corresponding vector $\alpha_k \nabla f(x_k)$ of its nonincremental counterpart differ by an error that is proportional to the stepsize (see the discussion in Exercises 2.6 and 2.10). For this reason a diminishing stepsize is essential for convergence to a minimizing point of f. However, it turns out that a peculiar form of convergence also typically occurs for the incremental gradient method if the stepsize α_k is a constant but sufficiently small α. In this case, the iterates converge to a "limit cycle," whereby the ith iterates ψ_i within the cycles converge to a different limit than the jth iterates ψ_j for $i \neq j$. The sequence $\{x_k\}$ of the iterates obtained at the end of cycles converges, except that the limit obtained *need not be optimal* even if f is convex. The limit tends to be close to an optimal point when the constant stepsize is small [for analysis of the case where the components f_i are quadratic, see Exercise 2.13(a), [BeT96] (Section 3.2), and [Ber99] (Section 1.5), where a linear convergence rate is also shown].

In practice, it is common to use a constant stepsize for a (possibly prespecified) number of iterations, then decrease the stepsize by a certain factor, and repeat, up to the point where the stepsize reaches a prespecified minimum. An alternative possibility is to use a stepsize α_k that diminishes to 0 at an appropriate rate [cf. Eq. (2.15)]. In this case convergence can be shown under reasonable conditions; see Exercise 2.10.

Still another possibility is to use an adaptive stepsize rule, whereby the stepsize is reduced (or increased) when the progress of the method indicates that the algorithm is oscillating because it operates within (or outside, respectively) the region of confusion. There are formal ways to implement such stepsize rules with sound convergence properties (see [Gri94], [Tse98], [MYF03]). One of the ideas is to look at a batch of incremental updates $\psi_i, \ldots, \psi_{i+M}$, for some relatively large $M \leq m$, and compare $\|\psi_i - \psi_{i+M}\|$ with $\sum_{\ell=1}^{M} \|\psi_{1+\ell-1} - \psi_{i+\ell}\|$. If the ratio of these two numbers is "small" this suggests that the method is oscillating.

Incremental gradient and subgradient methods have a rich theory, which includes convergence and rate of convergence analysis, optimization and randomization issues of the component order selection, and distributed computation aspects. Moreover they admit interesting combinations with other methods, such as the proximal algorithm. We will more fully discuss their properties and extensions in Chapter 6, Section 6.4.

Aggregated Gradient Methods

Another variant of incremental gradient is the *incremental aggregated gradient method*, which has the form

$$x_{k+1} = P_X\left(x_k - \alpha_k \sum_{\ell=0}^{m-1} \nabla f_{i_{k-\ell}}(x_{k-\ell})\right), \tag{2.35}$$

where f_{i_k} is the new component function selected for iteration k. Here, the component indexes i_k may either be selected in a cyclic order [$i_k = (k \text{ modulo } m) + 1$], or according to some randomization scheme, consistently with Eq. (2.31). Also for $k < m$, the summation should go up to $\ell = k$, and α should be replaced by a corresponding larger value, such as $\alpha_k = m\alpha/(k+1)$. This method, first proposed in [BHG08], computes the gradient incrementally, one component per iteration, but in place of the single component gradient, it uses an approximation to the total cost gradient $\nabla f(x_k)$, which is the sum of the component gradients computed in the past m iterations.

There is analytical and experimental evidence that by aggregating the component gradients one may be able to attain a faster asymptotic convergence rate, by ameliorating the effect of approximating the full gradient with component gradients; see the original paper [BHG08], which provides an analysis for quadratic problems, the paper [SLB13], which provides a more general convergence and convergence rate analysis, and extensive computational results, and the papers [Mai13], [Mai14], [DCD14], which describe related methods. The expectation of faster convergence should be tempered, however, because in order for the effect of aggregating the component gradients to fully manifest itself, at least one pass (and possibly quite a few more) through the components must be made, which may be too long if m is very large.

A drawback of this aggregated gradient method is that it requires that the most recent component gradients be kept in memory, so that when a component gradient is reevaluated at a new point, the preceding gradient of the same component is discarded from the sum of gradients of Eq. (2.35). There have been alternative implementations of the incremental aggregated gradient method idea that ameliorate this memory issue, by recalculating the full gradient periodically and replacing an old component gradient by a new one, once it becomes available; see [JoZ13], [ZMJ13], [XiZ14]. For an example, instead of the gradient sum

$$s_k = \sum_{\ell=0}^{m-1} \nabla f_{i_{k-\ell}}(x_{k-\ell}),$$

in Eq. (2.35), such a method may use $\tilde{s}_k$, updated according to

$$\tilde{s}_k = \nabla f_{i_k}(x_k) - \nabla f_{i_k}(\tilde{x}_k) + \tilde{s}_{k-1},$$

where $\tilde{s}_0$ is the full gradient computed at the start of the current cycle, and $\tilde{x}_k$ is the point at which this full gradient has been calculated. Thus to obtain $\tilde{s}_k$ one only needs to compute the difference of the two gradients

$$\nabla f_{i_k}(x_k) - \nabla f_{i_k}(\tilde{x}_k)$$

and add it to the current approximation of the full gradient $\tilde{s}_{k-1}$. This bypasses the need for extensive memory storage, and with proper implementation, typically leads to small degradation in performance. In particular, convergence with a sufficiently small constant stepsize, with an attendant superior convergence rate over the incremental gradient method, has been shown.

Incremental Gradient Method with Momentum

There is an incremental version of the gradient method with momentum or heavy ball method, discussed in Section 2.1.1 [cf. Eq. (2.12)]. It is given by

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) + \beta_k (x_k - x_{k-1}), \tag{2.36}$$

where f_{i_k} is the component function selected for iteration k, β_k is a scalar in $[0, 1)$, and we define $x_{-1} = x_0$; see e.g., [MaS94], [Tse98]. As noted earlier, special nonincremental methods with similarities to the one above have optimal iteration complexity properties under certain conditions; cf. Section 6.2. However, there have been no proposals of incremental versions of these optimal complexity methods.

The heavy ball method (2.36) is related with the aggregated gradient method (2.35) when $\beta_k \approx 1$. In particular, when $\alpha_k \equiv \alpha$ and $\beta_k \equiv \beta$, the sequence generated by Eq. (2.36) satisfies

$$x_{k+1} = x_k - \alpha \sum_{\ell=0}^{k} \beta^\ell \nabla f_{i_{k-\ell}}(x_{k-\ell}) \tag{2.37}$$

[both iterations (2.35) and (2.37) involve different types of diminishing dependence on past gradient components]. Thus, the heavy ball iteration (2.36) provides an approximate implementation of the incremental aggregated gradient method (2.35), while it does not have the memory storage issue of the latter.

A further way to intertwine the ideas of the aggregated gradient method (2.35) and the heavy ball method (2.36) for the unconstrained case ($X = \Re^n$) is to form an infinite sequence of components

$$f_1, f_2, \ldots, f_m, f_1, f_2, \ldots, f_m, f_1, f_2, \ldots, \tag{2.38}$$

and group together blocks of successive components into batches. One way to implement this idea is to add p preceding gradients (with $1 < p < m$) to the current component gradient in iteration (2.36), thus iterating according to

$$x_{k+1} = x_k - \alpha_k \sum_{\ell=0}^{p} \nabla f_{i_{k-\ell}}(x_{k-\ell}) + \beta_k (x_k - x_{k-1}). \tag{2.39}$$

Here f_{i_k} is the component function selected for iteration k using the order of the sequence (2.38). This essentially amounts to reformulating the problem by redefining the components as sums of $p+1$ successive components and applying an approximation of the incremental heavy ball method (2.36). The advantage of the method (2.39) over the aggregated gradient method is that it requires keeping in memory only p previous component gradients, and p can be chosen according to the memory limitations of the given computational environment. Generally in incremental methods, grouping together several components f_i, a process sometimes called *batching*, tends to reduce the size of the region of confusion (cf. Fig. 2.1.11), and with a small region of confusion, the incentive for aggregating the component gradients diminishes (see [Ber97] and [FrS12], for different implementations and analysis of this idea). The process of batching can also be implemented adaptively, based on some form of heuristic detection that the method has entered the region of confusion.

Stochastic Subgradient Methods

Incremental subgradient methods are related to methods that aim to minimize an expected value

$$f(x) = E\big\{F(x, w)\big\},$$

where w is a random variable, and $F(\cdot, w) : \Re^n \mapsto \Re$ is a convex function for each possible value of w. The stochastic subgradient method for minimizing f over a closed convex set X is given by

$$x_{k+1} = P_X\big(x_k - \alpha_k g(x_k, w_k)\big), \tag{2.40}$$

where w_k is a sample of w and $g(x_k, w_k)$ is a subgradient of $F(\cdot, w_k)$ at x_k. This method has a rich theory and a long history, particularly for the case where $F(\cdot, w)$ is differentiable for each value of w (for representative references, see [PoT73], [Lju77], [KuC78], [TBA86], [Pol87], [BeT89a], [BeT96], [Pfl96], [LBB98], [BeT00], [KuY03], [Bot05], [BeL07], [Mey07], [Bor08], [BBG09], [Ben09], [NJL09], [Bot10], [BaM11], [DHS11], [ShZ12], [FrG13], [NSW14]). It is strongly related to the classical algorithmic field of *stochastic approximation*; see the books [KuC78], [BeT96], [KuY03], [Spa03], [Mey07], [Bor08], [BPP13].

If we view the expected value cost $E\{F(x, w)\}$ as a weighted sum of cost function components, we see that the stochastic subgradient method (2.40) is related to the incremental subgradient method

$$x_{k+1} = P_X(x_k - \alpha_k g_{i,k}) \tag{2.41}$$

for minimizing a finite sum $\sum_{i=1}^m f_i$, when randomization is used for component selection [cf. Eq. (2.31)]. An important difference is that the former method involves sequential sampling of cost components $F(x, w)$ from an infinite population under some statistical assumptions, while in the latter the set of cost components f_i is predetermined and finite. However, it is possible to view the incremental subgradient method (2.41), with uniform randomized selection of the component function f_i (i.e., with i_k chosen to be any one of the indexes $1, \ldots, m$, with equal probability $1/m$, and independently of preceding choices), as a stochastic subgradient method.

Despite the apparent similarity of the incremental and the stochastic subgradient methods, the view that the problem

$$\begin{aligned} &\text{minimize} \quad f(x) = \sum_{i=1}^m f_i(x) \\ &\text{subject to} \quad x \in X, \end{aligned} \tag{2.42}$$

can simply be treated as a special case of the problem

$$\begin{aligned} &\text{minimize} \quad f(x) = E\{F(x, w)\} \\ &\text{subject to} \quad x \in X, \end{aligned}$$

is questionable.

One reason is that once we convert the finite sum problem to a stochastic problem, we preclude the use of methods that exploit the finite sum structure, such as the aggregated gradient methods we discussed earlier. Under certain conditions, these methods offer more attractive convergence rate guarantees than incremental and stochastic gradient methods, and can be very effective for many problems, as we have noted.

Another reason is that the finite-component problem (2.42) is often genuinely deterministic, and to view it as a stochastic problem at the outset

may mask some of its important characteristics, such as the number m of cost components, or the sequence in which the components are ordered and processed. These characteristics may potentially be algorithmically exploited. For example, with insight into the problem's structure, one may be able to discover a special deterministic or partially randomized order of processing the component functions that is superior to a uniform randomized order.

Example 2.1.6:

Consider the one-dimensional problem

$$\begin{aligned} &\text{minimize} \quad f(x) = \frac{1}{2}\sum_{i=1}^{m}(x - w_i)^2 \\ &\text{subject to} \quad x \in \Re, \end{aligned}$$

where the scalars w_i are given by

$$w_i = \begin{cases} 1 & \text{if } i\text{: odd,} \\ -1 & \text{if } i\text{: even.} \end{cases}$$

Assuming that m is an even number, the optimal solution is $x^* = 0$.

An incremental gradient method with the commonly used diminishing stepsize $\alpha_k = 1/(k+1)$ chooses a component index i_k at iteration k, and updates x_k according to

$$x_{k+1} = x_k - \frac{1}{k+1}(x_k - w_{i_k}),$$

starting with some initial iterate x_0. It is then easily verified by induction that

$$x_k = \frac{x_0}{k} + \frac{w_{i_0} + \cdots + w_{i_{k-1}}}{k}, \qquad k = 1, 2, \ldots.$$

Thus the iteration error, which is x_k (since $x^* = 0$), consists of two terms. The first is the error term x_0/k, which is independent of the method of selecting i_k, and the second is the error term

$$e_k = \frac{w_{i_0} + \cdots + w_{i_{k-1}}}{k},$$

which depends on the selection method for i_k.

If i_k is chosen by independently randomizing with equal probability $1/2$ over the odd and even cost components, then e_k will be a random variable whose variance can be calculated to be $1/2k$. Thus the standard deviation of the error x_k will be of order $O(1/\sqrt{k})$. If on the other hand i_k is chosen by the deterministic order, which alternates between the odd and even components, we will have $e_k = 1/k$ for the odd iterations and $e_k = 0$ for the even iterations, so the error x_k will be of order $O(1/k)$, much smaller than the one for the randomized order. Of course, this is a favorable deterministic order, and we

may obtain much worse results with an unfavorable deterministic order (such as selecting first all the odd components and then all the even components). However, the point here is that if we take the view that we are minimizing an expected value, we are disregarding at the outset information about the problem's structure that could be algorithmically useful.

A related experimental observation is that by suitably mixing the deterministic and the stochastic order selection methods we may produce better practical results. As an example, a popular technique for incremental methods, called *random reshuffling*, is to process the component functions f_i in cycles, with each component selected once in each cycle, and to reorder randomly the components after each cycle. This alternative order selection scheme has the nice property of allocating exactly one computation slot to each component in an m-slot cycle (m incremental iterations). By comparison, choosing components by uniform sampling allocates one computation slot to each component *on the average*, but some components may not get a slot while others may get more than one. A nonzero variance in the number of slots that any fixed component gets within a cycle, may be detrimental to performance, and suggests that reshuffling randomly the order of the component functions after each cycle works better. While it seems difficult to establish this fact analytically, a justification is suggested by the view of the incremental gradient method as a gradient method with error in the computation of the gradient (see Exercise 2.10). The error has apparently greater variance in the uniform sampling method than in the random reshuffling method. Heuristically, if the variance of the error is larger, the direction of descent deteriorates, suggesting slower convergence. For some experimental evidence, see [Bot09], [ReR13].

Let us also note that in Section 6.4 we will compare more formally various component selection orders in incremental methods. Our analysis will indicate that in the absence of problem-specific knowledge that can be exploited to select a favorable deterministic order, a uniform randomized order (each component f_i chosen with equal probability $1/m$ at each iteration, independently of preceding choices) has superior worst-case complexity.

Our conclusion is that in incremental methods, it may be beneficial to search for a favorable order for processing the component functions f_i, exploiting whatever problem-specific information may be available, rather than ignore all prior information and apply a uniform randomized order of the type commonly used in stochastic gradient methods. However, if a favorable order cannot be found, a randomized order is usually better than a fixed deterministic order, although there is no guarantee that this will be so for a given practical problem; for example a fixed deterministic order has been reported to be considerably faster on some benchmark test problems without any attempt to order the components favorably [Bot09].

Incremental Newton Methods

We will now consider an incremental version of Newton's method for unconstrained minimization of an additive cost function of the form

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

where the functions $f_i : \Re^n \mapsto \Re$ are convex and twice continuously differentiable. Consider the quadratic approximation $\tilde{f}_i$ of a function f_i at a vector $\psi \in \Re^n$, i.e., the second order Taylor expansion of f_i at ψ:

$$\tilde{f}_i(x;\psi) = \nabla f_i(\psi)'(x-\psi) + \tfrac{1}{2}(x-\psi)'\nabla^2 f_i(\psi)(x-\psi), \quad \forall\, x, \psi \in \Re^n.$$

Similar to Newton's method, which minimizes a quadratic approximation at the current point of the cost function [cf. Eq. (2.14)], the incremental form of Newton's method minimizes a sum of quadratic approximations of components. Similar to the incremental gradient method, we view an iteration as a cycle of m subiterations, each involving a single additional component f_i, and its gradient and Hessian at the current point within the cycle. In particular, if x_k is the vector obtained after k cycles, the vector x_{k+1} obtained after one more cycle is

$$x_{k+1} = \psi_{m,k},$$

where starting with $\psi_{0,k} = x_k$, we obtain $\psi_{m,k}$ after the m steps

$$\psi_{i,k} \in \arg\min_{x\in\Re^n} \sum_{\ell=1}^{i} \tilde{f}_\ell(x;\psi_{\ell-1,k}), \qquad i = 1,\ldots,m. \tag{2.43}$$

If all the functions f_i are quadratic, it can be seen that the method finds the solution in a single cycle.† The reason is that when f_i is quadratic, each $f_i(x)$ differs from $\tilde{f}_i(x;\psi)$ by a constant, which does not depend on x. Thus the difference

$$\sum_{i=1}^{m} f_i(x) - \sum_{i=1}^{m} \tilde{f}_i(x;\psi_{i-1,k})$$

† Here we assume that the m quadratic minimizations (2.43) to generate $\psi_{m,k}$ have a solution. For this it is sufficient that the first Hessian matrix $\nabla^2 f_1(x_0)$ be positive definite, in which case there is a unique solution at every iteration. A simple possibility to deal with this requirement is to add to f_1 a small positive definite quadratic term, such as $\frac{\epsilon}{2}\|x - x_0\|^2$. Another possibility is to lump together several of the component functions (enough to ensure that the sum of their quadratic approximations at x_0 is positive definite), and to use them in place of f_1. This is generally a good idea and leads to smoother initialization, as it ensures a relatively stable behavior of the algorithm for the initial iterations.

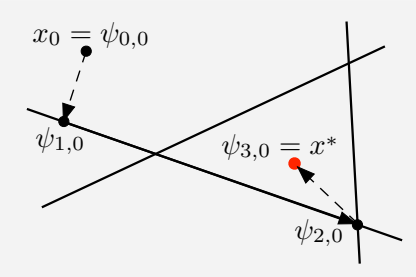

Figure 2.1.12. Illustration of the incremental Newton method for the case of a two-dimensional linear least squares problem with $m = 3$ cost function components (compare with the Kaczmarz method, cf. Fig. 2.1.10).

is a constant that is independent of x, and minimization of either sum in the above expression gives the same result.

As an example, consider a linear least squares problem, where

$$f_i(x) = \tfrac{1}{2}(c_i'x - b_i)^2, \qquad i = 1, \ldots, m.$$

Then the ith subiteration within a cycle minimizes

$$\sum_{\ell=1}^{i} f_\ell(x),$$

and when $i = m$, the solution of the problem is obtained (see Fig. 2.1.12). This convergence behavior should be compared with the one for the Kaczmarz method (cf. Fig. 2.1.10).

It is important to note that the quadratic minimizations of Eq. (2.43) can be carried out efficiently. For simplicity, let as assume that $\tilde{f}_1(x;\psi)$ is a positive definite quadratic, so that for all i, $\psi_{i,k}$ is well defined as the unique solution of the minimization problem in Eq. (2.43). We will show that the incremental Newton method (2.43) can be implemented in terms of the incremental update formula

$$\psi_{i,k} = \psi_{i-1,k} - D_{i,k}\nabla f_i(\psi_{i-1,k}), \tag{2.44}$$

where $D_{i,k}$ is given by

$$D_{i,k} = \left(\sum_{\ell=1}^{i} \nabla^2 f_\ell(\psi_{\ell-1,k})\right)^{-1}, \tag{2.45}$$

and is generated iteratively as

$$D_{i,k} = \left(D_{i-1,k}^{-1} + \nabla^2 f_i(\psi_{i,k})\right)^{-1}. \tag{2.46}$$

Indeed, from the definition of the method (2.43), the quadratic function $\sum_{\ell=1}^{i-1} \tilde{f}_\ell(x;\psi_{\ell-1,k})$ is minimized by $\psi_{i-1,k}$ and its Hessian matrix is $D_{i-1,k}^{-1}$, so we have

$$\sum_{\ell=1}^{i-1} \tilde{f}_\ell(x;\psi_{\ell-1,k}) = \tfrac{1}{2}(x-\psi_{\ell-1,k})'D_{i-1,k}^{-1}(x-\psi_{\ell-1,k}) + \text{constant}.$$

Thus, by adding $\tilde{f}_i(x;\psi_{i-1,k})$ to both sides of this expression, we obtain

$$\begin{aligned}\sum_{\ell=1}^{i} \tilde{f}_\ell(x;\psi_{\ell-1,k}) &= \tfrac{1}{2}(x-\psi_{\ell-1,k})'D_{i-1,k}^{-1}(x-\psi_{\ell-1,k}) + \text{constant}\\ &\quad + \tfrac{1}{2}(x-\psi_{i-1,k})'\nabla^2 f_i(\psi_{i-1,k})(x-\psi_{i-1,k}) + \nabla f_i(\psi_{i-1,k})'(x-\psi_{i-1,k}).\end{aligned}$$

Since by definition $\psi_{i,k}$ minimizes this function, we obtain Eqs. (2.44)-(2.46).

The update formula (2.46) for the matrix $D_{i,k}$ can often be efficiently implemented by using convenient formulas for the inverse of the sum of two matrices. In particular, if f_i is given by

$$f_i(x) = h_i(a_i'x - b_i),$$

for some twice differentiable convex function $h_i : \Re \mapsto \Re$, vector a_i, and scalar b_i, we have

$$\nabla^2 f_i(\psi_{i-1,k}) = \nabla^2 h_i(\psi_{i-1,k})\, a_i a_i',$$

and the update formula (2.46) can be written as

$$D_{i,k} = D_{i-1,k} - \frac{D_{i-1,k}a_i a_i' D_{i-1,k}}{\nabla^2 h_i(\psi_{i-1,k})^{-1} + a_i' D_{i-1,k} a_i};$$

this is the well-known Sherman-Morrison formula for the inverse of the sum of an invertible matrix and a rank-one matrix (see the matrix inversion formula in Section A.1 of Appendix A).

We have considered so far a single cycle of the incremental Newton method. One algorithmic possibility for cycling through the component functions multiple times, is to simply create a larger set of components by concatenating multiple copies of the original set, that is, by forming what we refer to as *the extended set of components*

$$f_1, f_2, \ldots, f_m,\ f_1, f_2, \ldots, f_m,\ f_1, f_2, \ldots.$$

The incremental Newton method, when applied to the extended set, asymptotically resembles a scaled incremental gradient method with diminishing stepsize of the type described earlier. Indeed, from Eq. (2.45)], the matrix

$D_{i,k}$ diminishes roughly in proportion to $1/k$. From this it follows that the asymptotic convergence properties of the incremental Newton method are similar to those of an incremental gradient method with diminishing stepsize of order $O(1/k)$. Thus its convergence rate is slower than linear.

To accelerate the convergence of the method one may employ a form of restart, so that $D_{i,k}$ does not converge to 0. For example $D_{i,k}$ may be reinitialized and increased in size at the beginning of each cycle. For problems where f has a unique nonsingular minimum x^* [one for which $\nabla^2 f(x^*)$ is nonsingular], one may design incremental Newton schemes with restart that converge linearly to within a neighborhood of x^* (and even superlinearly if x^* is also a minimum of all the functions f_i, so there is no region of confusion). Alternatively, the update formula (2.46) may be modified to

$$D_{i,k} = \Big(\lambda_k D_{i-1,k}^{-1} + \nabla^2 f_\ell(\psi_{i,k})\Big)^{-1}, \tag{2.47}$$

by introducing a fading factor $\lambda_k \in (0,1)$, which can be used to accelerate the practical convergence rate of the method (see [Ber96] for an analysis of schemes where $\lambda_k \to 1$; in cases where λ_k is some constant $\lambda < 1$, linear convergence to within a neighborhood of the optimum may be shown).

The following example provides some insight regarding the behavior of the method when the cost function f has a very large number of cost components, as is the case when f is defined as the average of a very large number of random samples.

Example 2.1.7: (Infinite Number of Cost Components)

Consider the problem

$$\begin{aligned} &\text{minimize} \quad f(x) \stackrel{\text{def}}{=} \lim_{m\to\infty} \frac{1}{m}\sum_{i=1}^{m} F(x,w_i) \\ &\text{subject to} \quad x \in \Re^n, \end{aligned}$$

where $\{w_k\}$ is a given sequence from some set, and each function $F(\cdot,w_i) : \Re^n \mapsto \Re$ is positive semidefinite quadratic. We assume that f is well-defined (i.e., the limit above exists for each $x \in \Re^n$), and is a positive definite quadratic. This type of problem arises in linear regression models (cf. Example 1.3.1) involving an infinite amount of data that is obtained through random sampling.

The natural extension of the incremental Newton's method, applied to the infinite set of components $F(\cdot,w_1), F(\cdot,w_2), \ldots$, generates the sequence $\{x_k^*\}$ where

$$x_k^* \in \arg\min_{x\in\Re^n} f_k(x) \stackrel{\text{def}}{=} \frac{1}{k}\sum_{i=1}^{k} F(x,w_i).$$

Since f is positive definite and the same is true for f_k, when k is large enough, we have $x_k^* \to x^*$, where x^* is the minimum of f. The rate of convergence

is determined strictly by the rate at which the vectors x_k^* approach x^*, or equivalently by the rate at which f_k approaches f. It is impossible to achieve a faster rate of convergence with an algorithm that is nonanticipative in the sense that it uses just the first k cost components in the first k iterations.

By contrast, if we were to apply the natural extension of the incremental gradient method to this problem, the convergence rate could be much worse. There would be an error due to the difference $(x_k^* - x^*)$, but also an additional error due to the difference $(x_k^* - x_k)$ between x_k^* and the kth iterate x_k of the incremental gradient method, which is generally diminishing quite slowly, possibly more slowly than $(x_k^* - x^*)$. The same is true for other gradient-type methods based on incremental computations, including the aggregated gradient methods discussed earlier.

Incremental Newton Method with Diagonal Approximation

Generally, with proper implementation, the incremental Newton method is often substantially faster than the incremental gradient method, in terms of numbers of iterations (there are theoretical results suggesting this property for stochastic versions of the two methods; see the end-of-chapter references). However, in addition to computation of second derivatives, the incremental Newton method involves greater overhead per iteration due to matrix-vector calculations in Eqs. (2.44), (2.46), and (2.47), so it is suitable only for problems where n, the dimension of x, is relatively small.

One way to remedy in part this difficulty is to approximate $\nabla^2 f_i(\psi_{i,k})$ by a diagonal matrix, and recursively update a diagonal approximation of $D_{i,k}$ using Eqs. (2.46) or (2.47). One possibility, inspired by similar diagonal scaling schemes for nonincremental gradient methods, is to set to 0 the off-diagonal components of $\nabla^2 f_i(\psi_{i,k})$. In this case, the iteration (2.44) becomes a diagonally scaled version of the incremental gradient method, and involves comparable overhead per iteration (assuming the required diagonal second derivatives are easily computed). As an additional option, one may multiply the diagonal components with a stepsize parameter that is close to 1 and add a small positive constant (to bound them away from 0). Ordinarily, for the convex problems considered here, this method should require little experimentation with stepsize selection.

Incremental Newton Methods with Constraints

The incremental Newton method can also be adapted to constrained problems of the form

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{m} f_i(x) \\ &\text{subject to} \quad x \in X, \end{aligned}$$

where $f_i : \Re^n \mapsto \Re$ are convex, twice continuously differentiable convex functions. If X has a relatively simple form, such as upper and lower

bounds on the variables, one may use a two-metric implementation, such as the ones discussed earlier, whereby the matrix $D_{i,k}$ is partially diagonalized before it is applied to the iteration

$$\psi_{i,k} = P_X\big(\psi_{i-1,k} - D_{i,k}\nabla f_i(\psi_{i-1,k})\big),$$

[cf. Eqs. (2.21) and (2.44)].

For more complicated constraint sets of the form

$$X = \cap_{i=1}^m X_i,$$

where each X_i is a relatively simple component constraint set (such as a halfspace), there is another possibility. This is to apply an incremental projected Newton iteration, with projection on a single individual component X_i, i.e., an iteration of the form

$$\psi_{i,k} \in \arg\min_{\psi\in X_i} \big\{\nabla f_i(\psi_{i-1,k})'(\psi-\psi_{i-1,k}) + \tfrac{1}{2}(\psi-\psi_{i-1,k})'H_{i,k}(\psi-\psi_{i-1,k})\big\},$$

where

$$H_{i,k} = \sum_{\ell=1}^{i} \nabla^2 f_\ell(\psi_{\ell-1,k}).$$

Note that each component X_i can be relatively simple, in which case the quadratic optimization problem above may be simple despite the fact that $H_{i,k}$ is nondiagonal. Depending on the problem's special structure, one may also use efficient methods that pass information from the solution of one quadratic subproblem to the next.

A similar method may also be used for problems of the form

$$\begin{aligned} &\text{minimize} \quad R(x) + \sum_{i=1}^m f_i(x) \\ &\text{subject to} \quad x \in X = \cap_{i=1}^m X_i, \end{aligned}$$

where $R(x)$ is a regularization function that is a multiple of either the ℓ_1 or the ℓ_2 norm. Then the incremental projected Newton iteration takes the form

$$\begin{aligned} \psi_{i,k} \in \arg\min_{\psi\in X_i} \big\{&R(\psi) + \nabla f_i(\psi_{i-1,k})'(\psi - \psi_{i-1,k}) \\ &+ \tfrac{1}{2}(\psi - \psi_{i-1,k})'H_{i,k}(\psi - \psi_{i-1,k})\big\}. \end{aligned}$$

When X_i is a polyhedral set, this problem is a quadratic program.

The idea of incremental projection on constraint set components is complementary to the idea of using gradient and possibly Hessian information from single cost function components at each iteration, and will be discussed in more detail in Section 6.4.4, in the context of incremental subgradient and incremental proximal methods. Several variations are possible, whereby the cost function component and constraint set component selected at each iteration may be chosen according to special deterministic or randomized rules. We refer to the papers [Ned11], [Ber11], [WaB13a] for a discussion of these incremental methods, their variations, and their convergence analysis.

Incremental Gauss-Newton Method – The Extended Kalman Filter

We will next consider an algorithm that operates similar to the incremental Newton method, but is specialized for the nonlinear least squares problem

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{m} \big\|g_i(x)\big\|^2 \\ &\text{subject to} \quad x \in \Re^n, \end{aligned}$$

where $g_i : \Re^n \mapsto \Re^{n_i}$ are some possibly nonlinear functions (cf. Example 1.3.1). As noted in Section 1.3, this is a common problem in practice.

We introduce a function $\tilde{g}_i$ that represents a linear approximation of g_i at a vector $\psi \in \Re^n$:

$$\tilde{g}_i(x;\psi) = \nabla g_i(\psi)'(x-\psi) + g_i(\psi), \qquad \forall\, x, \psi \in \Re^n,$$

where $\nabla g_i(\psi)$ is the $n \times n_i$ gradient matrix of g_i evaluated at ψ. Similar to the incremental gradient and Newton methods, we view an iteration as a cycle of m subiterations, each requiring linearization of a single additional component at the current point within the cycle. In particular, if x_k is the vector obtained after k cycles, the vector x_{k+1} obtained after one more cycle is

$$x_{k+1} = \psi_{m,k}, \tag{2.48}$$

where starting with $\psi_{0,k} = x_k$, we obtain $\psi_{m,k}$ after the m steps

$$\psi_{i,k} \in \arg\min_{x \in \Re^n} \sum_{\ell=1}^{i} \big\|\tilde{g}_\ell(x;\psi_{\ell-1,k})\big\|^2, \qquad i = 1, \ldots, m. \tag{2.49}$$

If all the functions g_i are linear, we have $\tilde{g}_i(x;\psi) = g_i(x)$, and the method solves the problem exactly in a single cycle. It then becomes identical to the incremental Newton method.

When the functions g_i are nonlinear the algorithm differs from the incremental Newton method because it does not involve second derivatives of g_i. It may be viewed instead as an incremental version of the Gauss-Newton method, a classical nonincremental scaled gradient method for solving nonlinear least squares problems (see e.g., [Ber99], Section 1.5). It is also known as the *extended Kalman filter*, and has found extensive application in state estimation and control of dynamic systems, where it was introduced in the mid-60s (it was also independently proposed in [Dav76]).

The implementation issues of the extended Kalman filter are similar to the ones of the incremental Newton method. This is because both methods solve similar linear least squares problems at each iteration [cf. Eqs. (2.43) and (2.49)]. The convergence behaviors of the two methods are

also similar: they asymptotically operate as scaled forms of incremental gradient methods with diminishing stepsize. Both methods are primarily well-suited for problems where the dimension of x is much smaller than the number of components in the additive cost function, so that the associated matrix-vector operations are not overly costly. Moreover their practical convergence rate can be accelerated by introducing a fading factor [cf. Eq. (2.47)]. We refer to [Ber96], [MYF03] for convergence analysis, variations, and computational experimentation.

2.1.6 Distributed Asynchronous Iterative Algorithms

We will now consider briefly distributed asynchronous counterparts of some of the algorithms discussed earlier in this section. We have in mind a situation where an iterative algorithm, such as a gradient method or a coordinate descent method, is parallelized by separating it into several local algorithms operating concurrently at different processors. The main characteristic of an asynchronous algorithm is that the local algorithms do not have to wait at predetermined points for predetermined information to become available. We thus allow some processors to execute more iterations than others, we allow some processors to communicate more frequently than others, and we allow the communication delays to be substantial and unpredictable.

Let us consider for simplicity the problem of unconstrained minimization of a differentiable function $f : \Re^n \mapsto \Re$. Out of the iterative algorithms of Sections 2.1.1-2.1.3, there are three types that are suitable for asynchronous distributed computation. Their asynchronous versions are as follows:

(a) *Gradient methods*, where we assume that the ith coordinate x^i is updated at a subset of times $\mathcal{R}_i \subset \{0, 1, \ldots\}$, according to

$$x_{k+1}^i = \begin{cases} x_k^i & \text{if } k \notin \mathcal{R}_i, \\ x_k^i - \alpha_k \dfrac{\partial f\left(x^1_{\tau_{i1}(k)}, \ldots, x^n_{\tau_{in}(k)}\right)}{\partial x^i} & \text{if } k \in \mathcal{R}_i, \end{cases} \qquad i = 1, \ldots, n,$$

where α_k is a positive stepsize. Here $\tau_{ij}(k)$ is the time at which the jth coordinate used in this update was computed, and the difference $k - \tau_{ij}(k)$ is commonly called the *communication delay* from j to i at time k. In a distributed setting, each coordinate x^i (or block of coordinates) may be updated by a separate processor, on the basis of values of coordinates made available by other processors, with some delay.

(b) *Coordinate descent methods*, where for simplicity we consider a block size of one; cf. Eq. (2.22). We assume that the ith scalar coordinate

is updated at a subset of times $\mathcal{R}_i \subset \{0, 1, \ldots\}$, according to

$$x_{k+1}^i \in \arg\min_{\xi \in \Re} f\left(x_{\tau_{i1}(k)}^1, \ldots, x_{\tau_{i,i-1}(k)}^{i-1}, \xi, x_{\tau_{i,i+1}(k)}^{i+1}, \ldots, x_{\tau_{in}(k)}^n\right),$$

and is left unchanged ($x_{k+1}^i = x_k^i$) if $k \notin \mathcal{R}_i$. The meanings of the subsets of updating times $\mathcal{R}_i$ and indexes $\tau_{ij}(k)$ are the same as in the case of gradient methods. Also the distributed environment where the method can be applied is similar to the case of the gradient method. Another practical setting that may be modeled well by this iteration is when all computation takes place at a single computer, but any number of coordinates may be simultaneously updated at a time, with the order of coordinate selection possibly being random.

(c) *Incremental gradient methods* for the case where

$$f(x) = \sum_{i=1}^m f_i(x).$$

Here the ith component is used to update x at a subset of times $\mathcal{R}_i$:

$$x_{k+1} = x_k - \alpha_k \nabla f_i\left(x_{\tau_{i1}(k)}^1, \ldots, x_{\tau_{in}(k)}^n\right), \qquad k \in \mathcal{R}_i,$$

where we assume that a single component gradient ∇f_i is used at each time (i.e., $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for $i \neq j$). The meaning of $\tau_{ij}(k)$ is the same as in the preceding cases, and the gradient ∇f_i can be replaced by a subgradient in the case of nondifferentiable f_i. Here the entire vector x is updated at a central computer, based on component gradients ∇f_i that are computed at other computers and are communicated with some delay to the central computer. For validity of these methods, it is essential that all the components f_i are used in the iteration with the same asymptotic frequency, $1/m$ (see [NBB01]). For this type of asynchronous implementation to make sense, the computation of ∇f_i must be substantially more time-consuming than the update of x_k using the preceding incremental iteration.

An interesting fact is that some asynchronous algorithms, called *totally asynchronous*, can tolerate arbitrarily large delays $k - \tau_{ij}(k)$, while other algorithms, called *partially asynchronous*, are not guaranteed to work unless there is an upper bound on these delays. The convergence mechanisms at work in each of these two cases are genuinely different and so are their analyses (see [BeT89a], where totally and partially asynchronous algorithms, and various special cases including gradient and coordinate descent methods, are discussed in Chapters 6 and 7, respectively).

The totally asynchronous algorithms are valid only under special conditions, which guarantee that any progress in the computation of the individual processors, is consistent with progress in the collective computation. For example to show convergence of the (synchronous) stationary iteration

$$x_{k+1} = G(x_k)$$

it is sufficient to show that G is a contraction mapping with respect to some norm (see Section A.4 of Appendix A), but for asynchronous convergence it turns out that one needs the contraction to be with respect to the sup-norm $\|x\|_\infty = \max_{i=1,\ldots,n} |x^i|$ or a weighted sup-norm (see Section 6.5.2). To guarantee totally asynchronous convergence of a gradient method with a constant and sufficiently small stepsize $\alpha_k \equiv \alpha$, a diagonal dominance condition is required; see the paper [Ber83]. In the special case of a quadratic cost function

$$f(x) = \tfrac{1}{2} x'Qx + b'x$$

this condition is that the Hessian matrix Q is diagonally dominant, i.e., has components q_{ij} such that

$$q_{ii} > \sum_{j \neq i} |q_{ij}|, \qquad i = 1, \ldots, n.$$

Without this diagonal dominance condition, totally asynchronous convergence is unlikely to be guaranteed (for examples see [BeT89a], Section 6.3.2).

The partially asynchronous algorithms do not need a weighted sup-norm contraction structure, but typically require either a diminishing or a stepsize that is small and inversely proportional to the size of the delays. The idea is that when the delays are bounded and the stepsize is small enough, the asynchronous algorithm resembles its synchronous counterpart sufficiently closely, so that the convergence properties of the latter are maintained (see [BeT89a], particularly Sections 7.1 and 7.5; also the convergence analysis of gradient methods with errors in the exercises). This mechanism for convergence is similar to the one for incremental methods. For this reason, incremental gradient, gradient projection, and coordinate descent methods are natural candidates for partially asynchronous implementation; see [BeT89a], Chapter 7, [BeT91], and [NBB01].

For further discussion of the implementation and convergence analysis of asynchronous algorithms, we refer to the survey [BeT91], the papers [TBA86], [SaB96], the books [BeT89a] (Chapters 6 and 7) and [Bor08] for deterministic and stochastic gradient, and coordinate descent methods, and the paper [NBB01] for incremental gradient and subgradient methods. For recent related work on distributed partially asynchronous algorithms of the gradient and coordinate descent type, see [NeO09a], [RRW11], and for partially asynchronous implementations of the Kaczmarz method, see [LWS14].

2.2 APPROXIMATION METHODS

Approximation methods for minimizing a convex function $f : \Re^n \mapsto \Re$ over a convex set X, are based on replacing f and X with approximations F_k

and X_k, respectively, at each iteration k, and finding

$$x_{k+1} \in \arg\min_{x \in X_k} F_k(x).$$

At the next iteration, F_{k+1} and X_{k+1} are generated by refining the approximation, based on the new point x_{k+1}, and possibly on the earlier points $x_k, \ldots, x_0$. Of course such a method makes sense only if the approximating problems are simpler than the original. There is a great variety of approximation methods, with different aims, and suitable for different circumstances. The present section provides a brief overview and orientation, while Chapters 4-6 provide a detailed analysis.

2.2.1 Polyhedral Approximation

In polyhedral approximation methods, F_k is a polyhedral function that approximates f and X_k is a polyhedral set that approximates X. The idea is that the approximate problem is polyhedral, so it may be easier to solve than the original problem. The methods include mechanisms for progressively refining the approximation, thereby obtaining a solution of the original problem in the limit. In some cases, only one of f and X is polyhedrally approximated.

In Chapter 4, we will discuss the two main approaches for polyhedral approximation: *outer linearization* (also called the *cutting plane* approach) and *inner linearization* (also called the *simplicial decomposition* approach). As the name suggests, outer linearization approximates $\text{epi}(f)$ and X from without, $F_k(x) \leq f(x)$ for all x, and $X_k \supset X$, using intersections of finite numbers of halfspaces. By contrast, inner linearization approximates $\text{epi}(f)$ and X from within, $F_k(x) \geq f(x)$ for all x, and $X_k \subset X$, using convex hulls of finite numbers of halflines or points. Figure 2.2.1 illustrates outer and inner linearization of convex sets and functions.

We will show in Sections 4.3 and 4.4 that these two approaches are intimately connected by conjugacy and duality: *the dual of an outer approximating problem is an inner approximating problem involving the conjugates of F_k and the indicator function of X_k, and reversely.* In fact, using this duality, outer and inner approximations may be combined in the same algorithm.

One of the major applications of the cutting plane approach is in *Dantzig-Wolfe decomposition*, an important method for solving large scale problems with special structure, including the separable problems of Section 1.1.1 (see e.g., [BeT97], [Ber99]). Simplicial decomposition also finds many important applications in problems with special structure; e.g., in high-dimensional problems with a constraint set X such that minimization of a linear function over X is relatively simple. This is exactly the same structure that favors the use of the conditional gradient method discussed in Section 2.1.2 (see Chapter 4). A prominent example is the multicommodity flow problem of Example 1.4.5.

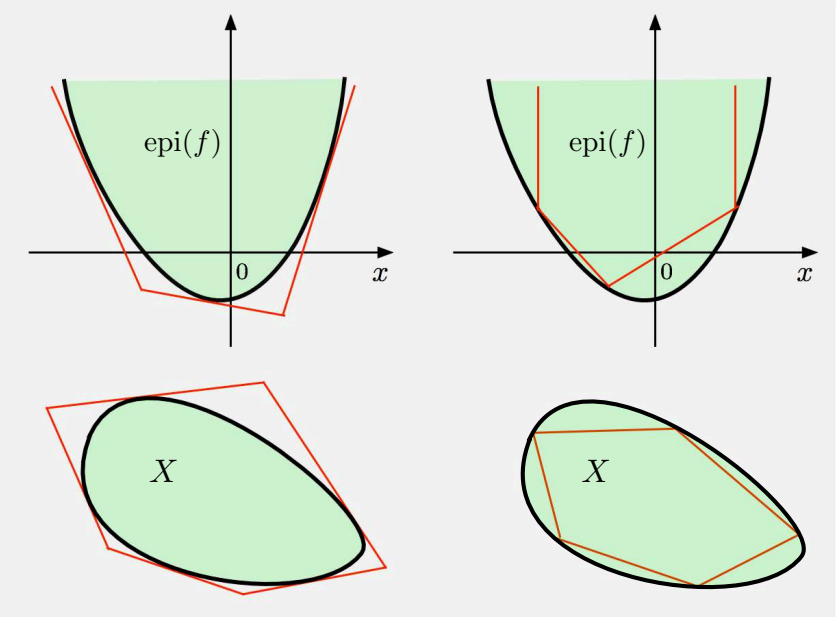

Figure 2.2.1. Illustration of outer and inner linearization of a convex function f and a convex set X using hyperplanes and convex hulls.

2.2.2 Penalty, Augmented Lagrangian, and Interior Point Methods

Generally in optimization problems, the presence of constraints complicates the algorithmic solution, and limits the range of available algorithms. For this reason it is natural to try to eliminate constraints by using approximation of the corresponding indicator functions. In particular, we may replace constraints by penalty functions that prescribe a high cost for their violation. We discussed in Section 1.5 such an approximation scheme, which uses exact nondifferentiable penalty functions. In this section we focus on differentiable penalty functions that are not necessarily exact.

To illustrate this approach, let us consider the equality constrained problem

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in X, \quad a_i' x = b_i, \quad i = 1, \ldots, m. \end{aligned} \tag{2.50}$$

We replace this problem with a penalized version

$$\begin{aligned} &\text{minimize} \quad f(x) + c_k \sum_{i=1}^{m} P(a_i' x - b_i) \\ &\text{subject to} \quad x \in X, \end{aligned} \tag{2.51}$$

where $P(\cdot)$ is a scalar penalty function satisfying

$$P(u) = 0 \quad \text{if} \quad u = 0,$$

and

$$P(u) > 0 \quad \text{if} \quad u \neq 0.$$

The scalar c_k is a positive penalty parameter, so by increasing c_k to ∞, the solution x_k of the penalized problem tends to decrease the constraint violation, thereby providing an increasingly accurate approximation to the original problem. An important practical point here is that c_k should be increased gradually, using the optimal solution of each approximating problem to start the algorithm that solves the next approximating problem. Otherwise serious numerical problems occur due to "ill-conditioning."

A common choice for P is the quadratic penalty function

$$P(u) = \tfrac{1}{2}u^2,$$

in which case the penalized problem (2.51) takes the form

$$\begin{aligned} &\text{minimize} \quad f(x) + \frac{c_k}{2}\|Ax - b\|^2 \\ &\text{subject to} \quad x \in X, \end{aligned} \tag{2.52}$$

where $Ax = b$ is a vector representation of the system of equations $a_i'x = b_i$, $i = 1, \ldots, m$.

An important enhancement of the penalty function approach is the augmented Lagrangian methodology, where we add a linear term to $P(u)$, involving a multiplier vector $\lambda_k \in \Re^m$. Then in place of problem (2.52), we solve the problem

$$\begin{aligned} &\text{minimize} \quad f(x) + \lambda_k'(Ax - b) + \frac{c_k}{2}\|Ax - b\|^2 \\ &\text{subject to} \quad x \in X. \end{aligned} \tag{2.53}$$

After a minimizing vector x_k is obtained, the multiplier vector λ_k is updated by some formula that aims to approximate an optimal dual solution. A common choice that we will discuss in Chapter 5 is

$$\lambda_{k+1} = \lambda_k + c_k(Ax_k - b). \tag{2.54}$$

This is also known as the *first order augmented Lagrangian method* (also called *first order method of multipliers*). It is a major general purpose, highly reliable, constrained optimization method, which applies to nonconvex problems as well. It has a rich theory, with a strong connection to duality, and many variations that are aimed at increased efficiency, involving for example second order multiplier updates and inexact minimization of the augmented Lagrangian. In the convex programming setting of

this book, augmented Lagrangian methods embody additional favorable structure. Among others, convergence is guaranteed for *any* nondecreasing sequence $\{c_k\}$ (for nonconvex problems, c_k must exceed a certain positive threshold). Moreover there is no requirement that $c_k \to \infty$, which is needed for penalty methods that do not involve multiplier updates, and is often the cause of numerical problems.

Generally, penalty and augmented Lagrangian methods can be used for inequality as well as equality constraints. The penalty function is modified to reflect penalization for violation of inequalities. For example the inequality constraint analog of the quadratic penalty $P(u) = \frac{1}{2}u^2$ is

$$P(u) = \tfrac{1}{2}\big(\max\{0, u\}\big)^2.$$

We will consider these possibilities in greater detail in Section 5.2.1.

The penalty methods just discussed are known as *exterior penalty methods*: they approximate the indicator function of the constraint set from without. Another type of algorithm involves approximation from within, which leads to the so called *interior point methods*. These are important methods that find application in a broad variety of problems, including linear programming. They will be discussed in Section 6.8.

2.2.3 Proximal Algorithm, Bundle Methods, and Tikhonov Regularization

The proximal algorithm, briefly discussed in Section 2.1.4, aims to minimize a closed proper convex function $f : \Re^n \mapsto (-\infty, \infty]$, and is given by

$$x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ f(x) + \frac{1}{2c_k}\|x - x_k\|^2 \right\}, \tag{2.55}$$

[cf. Eq. (2.25)], where x_0 is an arbitrary starting point and c_k is a positive scalar parameter. As the parameter c_k tends to ∞, the quadratic regularization term becomes insignificant and the proximal minimization (2.55) approximates more closely the minimization of f, hence the connection of the proximal algorithm with the approximation approach.

We will discuss the proximal algorithm in much more detail in Chapter 5, including dual and polyhedral approximation versions. Among others, we will show that when f is the dual function of the constrained optimization problem (2.50), the proximal algorithm, via Fenchel duality, becomes equivalent to the multiplier iteration of the augmented Lagrangian method [cf. Eq. (2.54)]. Since any closed proper convex function can be viewed as the dual function of an appropriate convex constrained optimization problem, it follows that *the proximal algorithm (2.55) is essentially equivalent to the augmented Lagrangian method*: the two algorithms are dual sides of the same coin.

There are also variants of the proximal algorithm where f in Eq. (2.55) is approximated by a polyhedral or other function. One possibility is *bundle methods*, which involve a combination of the proximal and polyhedral approximation ideas. The motivation here is to simplify the proximal minimization subproblem (2.25), replacing it for example with a quadratic programming problem. Some of these methods may be viewed as regularized versions of Dantzig-Wolfe decomposition (see Section 4.3).

Another approximation approach that bears similarity to the proximal algorithm is *Tikhonov regularization*, which approximates the minimization of f with the minimization

$$x_{k+1} \in \arg\min_{x \in \Re^n} \left\{ f(x) + \frac{1}{2c_k}\|x\|^2 \right\}. \tag{2.56}$$

The quadratic regularization term makes the cost function of the preceding problem strictly convex, and guarantees that it has a unique minimum. Sometimes the quadratic term in Eq. (2.56) is scaled and a term $\|Sx\|^2$ is used instead, where S is a suitable scaling matrix. The difference with the proximal algorithm (2.55) is that x_k does not enter directly the minimization to determine x_{k+1}, so the method relies for its convergence on increasing c_k to ∞. By contrast this is not necessary for the proximal algorithm, which is generally convergent even when c_k is left constant (as we will see in Section 5.1), and is typically much faster. Similar to the proximal algorithm, there is a dual and essentially equivalent algorithm to Tikhonov regularization. This is the penalty method that consists of sequential minimization of the quadratically penalized cost function (2.52) for a sequence $\{c_k\}$ with $c_k \to \infty$.

2.2.4 Alternating Direction Method of Multipliers

The proximal algorithm embodies fundamental ideas that lead to a variety of other interesting methods. In particular, when properly generalized (see Section 5.1.4), it contains as a special case the *alternating direction method of multipliers* (ADMM for short), a method that resembles the augmented Lagrangian method, but is well-suited for some important classes of problems with special structure.

The starting point for the ADMM is the minimization problem of the Fenchel duality context:

$$\begin{aligned} &\text{minimize} \quad f_1(x) + f_2(Ax) \\ &\text{subject to} \quad x \in \Re^n, \end{aligned} \tag{2.57}$$

where A is an $m \times n$ matrix, $f_1 : \Re^n \mapsto (-\infty, \infty]$ and $f_2 : \Re^m \mapsto (-\infty, \infty]$ are closed proper convex functions. We convert this problem to the equivalent constrained minimization problem

$$\begin{aligned} &\text{minimize} \quad f_1(x) + f_2(z) \\ &\text{subject to} \quad x \in \Re^n,\ z \in \Re^m, \quad Ax = z, \end{aligned} \tag{2.58}$$

and we introduce its augmented Lagrangian function

$$L_c(x, z, \lambda) = f_1(x) + f_2(z) + \lambda'(Ax - z) + \frac{c}{2}\|Ax - z\|^2,$$

where c is a positive parameter.

The ADMM, given the current iterates $(x_k, z_k, \lambda_k) \in \Re^n \times \Re^m \times \Re^m$, generates a new iterate $(x_{k+1}, z_{k+1}, \lambda_{k+1})$ by first minimizing the augmented Lagrangian with respect to x, then with respect to z, and finally performing a multiplier update:

$$x_{k+1} \in \arg\min_{x \in \Re^n} L_c(x, z_k, \lambda_k), \tag{2.59}$$

$$z_{k+1} \in \arg\min_{z \in \Re^m} L_c(x_{k+1}, z, \lambda_k), \tag{2.60}$$

$$\lambda_{k+1} = \lambda_k + c(Ax_{k+1} - z_{k+1}). \tag{2.61}$$

The important advantage that the ADMM may offer over the augmented Lagrangian method, is that it does not involve a joint minimization with respect to x and z. Thus the complications resulting from the coupling of x and z in the penalty term $\|Ax - z\|^2$ of the augmented Lagrangian are eliminated. This property can be exploited in special applications, for which the ADMM is structurally well suited, as we will discuss in Section 5.4. On the other hand the ADMM may converge more slowly than the augmented Lagrangian method, so the flexibility it provides must be weighted against this potential drawback.

In Chapter 5, we will see that the proximal algorithm for minimization can be viewed as a special case of a generalized proximal algorithm for finding a solution of an equation involving a multivalued monotone operator. While we will not fully develop the range of algorithms that are based on this generalization, we will show that both the augmented Lagrangian method and the ADMM are special cases of the generalized proximal algorithm, corresponding to two different multivalued monotone operators. Because of the differences of these two operators, some of the properties of the two methods are quite different. For example, contrary to the case of the augmented Lagrangian method (where c_k is often taken to be increasing with k in order to accelerate convergence), there seems to be no generally good way to adjust c in ADMM from one iteration to the next. Moreover, even when both methods have a linear convergence rate, the performance of the two methods may differ markedly in practice. Still there is more than a superficial connection between the two methods, which can be understood within the context of their common proximal algorithm ancestry.

In Section 6.3, we will also see another connection of ADMM with proximal-related methods, and particularly the proximal gradient method, which we briefly discussed in Section 2.1.4 [cf. Eq. (2.27)]. It turns out that both the ADMM and the proximal gradient method can be viewed

as instances of splitting algorithms for finding a zero of the sum of two monotone operators. The idea is to decouple the two operators within an iteration: one operator is treated by a proximal-type algorithm, while the other is treated by a proximal-type or a gradient algorithm. In so doing, the complications that arise from coupling of the two operators are mitigated.

2.2.5 Smoothing of Nondifferentiable Problems

Generally speaking, differentiable cost functions are preferable to nondifferentiable ones, because algorithms for the former are better developed and are more effective than algorithms for the latter. Thus there is an incentive to eliminate nondifferentiabilities by "smoothing" their corners. It turns out that penalty functions and smoothing are closely related, reflecting the fact that constraints and nondifferentiabilities are also closely related. As an example of this connection, the unconstrained minimax problem

$$\begin{aligned} &\text{minimize} \quad \max\big\{f_1(x),\ldots,f_m(x)\big\} \\ &\text{subject to} \quad x\in\Re^n, \end{aligned} \tag{2.62}$$

where $f_1,\ldots,f_m$ are differentiable functions can be converted to the differentiable constrained problem

$$\begin{aligned} &\text{minimize} \quad z \\ &\text{subject to} \quad f_j(x)\le z, \quad j=1,\ldots,m, \end{aligned} \tag{2.63}$$

where z is an artificial scalar variable. When a penalty or augmented Lagrangian method is applied to the constrained problem (2.63), we will show that a smoothing method is obtained for the minimax problem (2.62).

We will now describe a technique (first given in [Ber75b], and generalized in [Ber77]) to obtain smoothing approximations. Let $f:\Re^n\mapsto(-\infty,\infty]$ be a closed proper convex function with conjugate denoted by $f^\star$. For fixed $c>0$ and $\lambda\in\Re^n$, define

$$f_{c,\lambda}(x)=\inf_{u\in\Re^n}\left\{f(x-u)+\lambda' u+\frac{c}{2}\|u\|^2\right\},\qquad x\in\Re^n. \tag{2.64}$$

The conjugates of $\phi_1(u)=f(x-u)$ and $\phi_2(u)=\lambda' u+\frac{c}{2}\|u\|^2$ are $\phi_1^\star(y)=f^\star(-y)+y'x$ and $\phi_2^\star(y)=\frac{1}{2c}\|y-\lambda\|^2$, so by using the Fenchel duality formula $\inf_{u\in\Re^n}\big\{\phi_1(u)+\phi_2(u)\big\}=\sup_{y\in\Re^n}\big\{-\phi_1^\star(-y)-\phi_2^\star(y)\big\}$, we have

$$f_{c,\lambda}(x)=\sup_{y\in\Re^n}\left\{y'x-f^\star(y)-\frac{1}{2c}\|y-\lambda\|^2\right\},\qquad x\in\Re^n. \tag{2.65}$$

It can be seen that $f_{c,\lambda}$ approximates f in the sense that

$$\lim_{c\to\infty}f_{c,\lambda}(x)=f^{\star\star}(x)=f(x),\qquad\forall\ x,\lambda\in\Re^n;$$

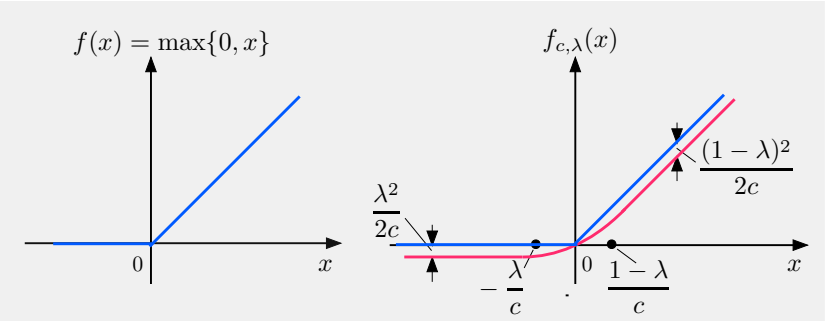

Figure 2.2.2. Illustration of smoothing of the function $f(x) = \max\{0, x\}$. Note that as $c \to \infty$, we have $f_{c,\lambda}(x) \to f(x)$ for all $x \in \Re$, regardless of the value of λ.

the double conjugate $f^{\star\star}$ is equal to f by the Conjugacy Theorem (Prop. 1.6.1 in Appendix B). Furthermore, it can be shown using the optimality conditions of the Fenchel Duality Theorem Prop. 1.2.1(c) (see also [Ber77]) that $f_{c,\lambda}$ is convex and differentiable as a function of x for fixed c and λ, and that the gradient $\nabla f_{c,\lambda}(x)$ at any $x \in \Re^n$ can be obtained in two ways:

(i) As the vector $\lambda + cu$, where u is the unique vector attaining the infimum in Eq. (2.64).

(ii) As the unique vector y that attains the supremum in Eq. (2.65).

The smoothing approach consists of replacing unsmoothed functions f with their smooth approximations $f_{c,\lambda}$, wherever they occur within the cost and constraint functions of a given problem. Note that there may be several functions f that are being smoothed simultaneously, and each occurrence of f may use different λ and c. In this way we obtain a differentiable problem that approximates the original.

An an example consider a common source of nondifferentiability:

$$f(x) = \max\{0, x\}, \qquad x \in \Re.$$

It can be verified using Eqs. (2.64) and (2.65) that

$$f_{c,\lambda}(x) = \begin{cases} x - \dfrac{(1-\lambda)^2}{2c} & \text{if } \frac{1-\lambda}{c} \leq x, \\ \lambda x + \dfrac{c}{2}x^2 & \text{if } -\frac{\lambda}{c} \leq x \leq \frac{1-\lambda}{c}, \\ -\dfrac{\lambda^2}{2c} & \text{if } x \leq -\frac{\lambda}{c}; \end{cases}$$

see Fig. 2.2.2. The function $f(x) = \max\{0, x\}$ may also be used as a building block to construct more complicated nondifferentiable functions, such as for example

$$\max\{x_1, x_2\} = x_1 + \max\{0, x_1 - x_2\};$$

see [Ber82a], Ch. 3.

Smoothing and Augmented Lagrangians

The smoothing technique just described can also be combined with the augmented Lagrangian method. As an example, let $f : \Re^n \mapsto (-\infty, \infty]$ be a closed proper convex function with conjugate denoted by $f^\star$. Let $F : \Re^n \mapsto \Re$ be another convex function, and let X be a closed convex set. Consider the problem

$$\begin{aligned} &\text{minimize} \quad F(x) + f(x) \\ &\text{subject to} \quad x \in X, \end{aligned}$$

and the equivalent problem

$$\begin{aligned} &\text{minimize} \quad F(x) + f(x-u) \\ &\text{subject to} \quad x \in X,\ u = 0. \end{aligned}$$

Applying the augmented Lagrangian method (2.53)-(2.54) to the latter problem leads to minimizations of the form

$$(x_{k+1}, u_{k+1}) \in \arg \min_{x \in X,\, u \in \Re^n} \left\{ F(x) + f(x-u) + \lambda_k' u + \frac{c_k}{2}\|u\|^2 \right\}.$$

By first minimizing over $u \in \Re^n$, these minimizations yield

$$x_{k+1} \in \arg\min_{x \in X} \left\{ F(x) + f_{c_k,\lambda_k}(x) \right\},$$

where f_{c_k,λ_k} is the smoothed function

$$f_{c_k,\lambda_k}(x) = \inf_{u \in \Re^n} \left\{ f(x-u) + \lambda_k' u + \frac{c_k}{2}\|u\|^2 \right\},$$

[cf. Eq. (2.64)]. The corresponding multiplier update (2.54) is

$$\lambda_{k+1} = \lambda_k + c_k u_{k+1},$$

where

$$u_{k+1} \in \arg \min_{u \in \Re^n} \left\{ f(x_{k+1} - u) + \lambda_k{}' u + \frac{c_k}{2}\|u\|^2 \right\}.$$

The preceding technique can be extended so that it applies to general convex/concave minimax problems. Let Z be a nonempty convex subset of $\Re^m$, respectively, and $\phi : \Re^n \times Z \mapsto \Re$ is a function such that $\phi(\cdot, z) : \Re^n \mapsto \Re$ is convex for each $z \in Z$, and $-\phi(x, \cdot) : Z \mapsto \Re$ is convex and closed for each $x \in \Re^n$. Consider the problem

$$\begin{aligned} &\text{minimize} \quad \sup_{z \in Z} \phi(x, z) \\ &\text{subject to} \quad x \in X, \end{aligned}$$

where X is a nonempty closed convex subset of $\Re^n$. Consider also the equivalent problem

$$\begin{aligned} &\text{minimize} \quad f(x,y) \\ &\text{subject to} \quad x \in X,\ y = 0, \end{aligned}$$

where f is the function

$$f(x,y) = \sup_{z \in Z} \big\{ \phi(x,z) - y'z \big\}, \qquad x \in \Re^n,\ y \in \Re^m,$$

which is closed and convex, being the supremum of closed and convex functions. The augmented Lagrangian minimization (2.53) for this problem takes the form

$$x_{k+1} \in \arg\min_{x \in X} f_{c_k,\lambda_k}(x),$$

where $f_{c,\lambda} : \Re^n \mapsto \Re$ is the differentiable function given by

$$f_{c,\lambda}(x) = \min_{y \in \Re^m} \left\{ f(x,y) + \lambda' y + \frac{c}{2}\|y\|^2 \right\}, \qquad x \in \Re^n.$$

The corresponding multiplier update (2.54) is

$$\lambda_{k+1} = \lambda_k + c_k y_{k+1},$$

where

$$y_{k+1} \in \arg\min_{y \in \Re^m} \left\{ f(x_{k+1},y) + \lambda_k' y + \frac{c_k}{2}\|y\|^2 \right\}.$$

This method of course makes sense only in the case where the function f has a convenient form that facilitates the preceding minimization.

For further discussion of the relations and combination of smoothing with the augmented Lagrangian method, see [Ber75b], [Ber77], [Pap81], and for a detailed textbook analysis, [Ber82a], Ch. 3. There have also been many variations of smoothing ideas and applications in different contexts; see [Ber73], [Geo77], [Pol79], [Pol88], [BeT89b], [PiZ94], [Nes05], [Che07], [OvG14]. In Section 6.2, we will also see an application of smoothing as an analytical device, in the context of complexity analysis.

Exponential Smoothing

We have used so far a quadratic penalty function as the basis for smoothing. It is also possible to use other types of penalty functions. A simple penalty function, which often leads to convenient formulas is the exponential, which will also be discussed further in Section 6.6. The advantage of the exponential function over the quadratic is that it produces twice differentiable approximating functions. This may be significant when Newton's method is used to solve the smoothed problem.

As an example, a smooth approximation of the function

$$f(x) = \max\big\{f_1(x), \ldots, f_m(x)\big\}$$

is given by

$$f_{c,\lambda}(x) = \frac{1}{c} \ln \left\{ \sum_{i=1}^{m} \lambda^i e^{c f_i(x)} \right\}, \tag{2.66}$$

where $c > 0$, and $\lambda = (\lambda^1, \ldots, \lambda^m)$ is a vector with

$$\sum_{i=1}^{m} \lambda^i = 1, \qquad \lambda^i > 0, \quad \forall\ i = 1, \ldots, m.$$

There is an augmented Lagrangian method associated with the approximation (2.66). It involves minimizing over $x \in \Re^n$ the function $f_{c_k,\lambda_k}(x)$ for a given c_k and λ_k to obtain an approximation x_k to the minimum of f. This approximation is refined by setting $c_{k+1} \geq c_k$ and

$$\lambda_{k+1}^i = \frac{\lambda_k^i e^{c_k f_i(x_k)}}{\sum_{j=1}^m \lambda_k^j e^{c_k f_j(x_k)}}, \qquad i = 1, \ldots, m, \tag{2.67}$$

and by repeating the process.† The generated sequence $\{x_k\}$ can be shown to converge to the minimum of f under mild assumptions, based on general convergence properties of augmented Lagrangian methods that use nonquadratic penalty functions; see [Ber82a], Ch. 5, for a detailed development.

Example 2.2.1: (Smoothed ℓ_1 Regularization)

Consider the ℓ_1-regularized least squares problem

$$\begin{aligned} &\text{minimize} \quad \gamma \sum_{j=1}^{n} |x^j| + \frac{1}{2} \sum_{i=1}^{m} (a_i' x - b_i)^2 \\ &\text{subject to} \quad x \in \Re^n, \end{aligned}$$

† Sometimes the use of the exponential in Eq. (2.67) and other related formulas, such as (2.66), may lead to very large numbers and computer overflow. In this case one may use a translation device to avoid such numbers, e.g., multiplying numerator and denominator in Eq. (2.67) by $e^{-\beta_k}$ where

$$\beta_k = \max_{i=1,\ldots,m} \big\{c_k f_i(x_k)\big\}.$$

A similar idea works for Eq. (2.66).

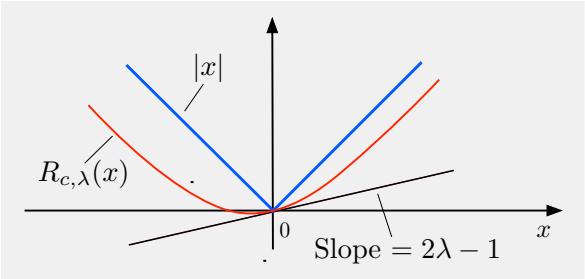

Figure 2.2.3. Illustration of the exponentially smoothed version

$$R_{c,\lambda}(x) = \frac{1}{c} \ln \left\{ \lambda e^{cx} + (1-\lambda) e^{-cx} \right\}$$

of the absolute value function $|x|$. The approximation becomes asymptotically exact as $c \to \infty$ for any fixed value of the multiplier $\lambda \in (0,1)$. Also by adjusting the multiplier λ within the range $(0,1)$, we can attain better approximation for x positive or negative. As $\lambda \to 1$ (or $\lambda \to 0$) the approximation becomes asymptotically exact for $x \geq 0$ (or $x \leq 0$, respectively).

where a_i and b_i are given vectors and scalars, respectively (cf. Example 1.3.1). The nondifferentiable ℓ_1 penalty may be smoothed by writing each term $|x^j|$ as $\max\{x^j, -x^j\}$ and by smoothing it using Eq. (2.66), i.e., replacing it by

$$R_{c,\lambda^j}(x^j) = \frac{1}{c} \ln \left\{ \lambda^j e^{cx^j} + (1-\lambda^j) e^{-cx^j} \right\},$$

where c and λ^j are scalars satisfying $c > 0$ and $\lambda^j \in (0,1)$ (see Fig. 2.2.3). We may then consider an exponential type of augmented Lagrangian method, whereby we minimize over $\Re^n$ the twice differentiable function

$$\gamma \sum_{j=1}^{n} R_{c_k,\lambda_k^j}(x^j) + \frac{1}{2} \sum_{i=1}^{m} (a_i' x - b_i)^2, \tag{2.68}$$

to obtain an approximation x_k to the optimal solution. This approximation is refined by setting $c_{k+1} \geq c_k$ and

$$\lambda_{k+1}^j = \frac{\lambda_k^j e^{c_k x_k^j}}{\lambda_k^j e^{c_k x_k^j} + (1-\lambda_k^j) e^{-c_k x_k^j}}, \qquad j = 1, \ldots, n, \tag{2.69}$$

[cf. Eq. (2.67)], and by repeating the process. Note that the minimization of the exponentially smoothed cost function (2.68) can be carried out efficiently by incremental methods, such as the incremental gradient and Newton methods of Section 2.1.5.

As Fig. 2.2.3 suggests, the adjustment of the multiplier λ^j can selectively reduce the error

$$|x^j| - R_{c,\lambda^j}(x^j)$$

depending on whether good approximation for positive or negative x^j is desired. For this reason *it is not necessary to increase c_k to infinity*; the multiplier iteration (2.69) is sufficient for convergence even with c_k kept constant at some positive value (see [Ber82a], Ch. 5).

2.3 NOTES, SOURCES, AND EXERCISES

Section 2.1: Textbooks on nonlinear programming with a substantial algorithmic content are good sources for the overview material on unconstrained and constrained differentiable optimization in this chapter, e.g., Zangwill [Zan69], Polak [Pol71], Hestenes [Hes75], Zoutendijk [Zou76], Shapiro [Sha79], Gill, Murray, and Wright [GMW81], Luenberger [Lue84], Poljak [Pol87], Dennis and Schnabel [DeS96], Bertsekas [Ber99], Kelley [Kel99], Fletcher [Fle00], Nesterov [Nes04], Bazaraa, Shetty, and Sherali [BSS06], Nocedal and Wright [NoW06], Ruszczynski [Rus06], Griva, Nash, and Sofer [GNS08], Luenberger and Ye [LuY08]. References for specific algorithmic nondifferentiable optimization topics will be given in subsequent chapters.

Incremental gradient methods have a long history, particularly for the unconstrained case ($X = \Re^n$), starting with the Widrow-Hoff least mean squares (LMS) method [WiH60], which stimulated much subsequent research. They have also been used widely, under the generic name "backpropagation methods," for training of neural networks, which involves nonquadratic/nonconvex differentiable cost components. There is an extensive literature on this subject, and for some representative works, we refer to the papers by Rumelhart, Hinton, and Williams [RHW86], [RHW88], Becker and LeCun [BeL88], Vogl et al. [VMR88], and Saarinen, Bramley, and Cybenko [SBC91], and the books by Bishop [Bis95], Bertsekas and Tsitsiklis [BeT96], and Haykin [Hay08]. Some of this literature overlaps with the literature on stochastic gradient methods, which we noted in Section 2.1.5.

Deterministic convergence analyses of several variants of incremental gradient methods were given in the 90s under various assumptions and for a variety of stepsize rules; see Luo [Luo91], Grippo [Gri94], [Gri00], Luo and Tseng [LuT94a], Mangasarian and Solodov [MaS94], Bertsekas [Ber97], Solodov [Sol98], Tseng [Tse98], and Bertsekas and Tsitsiklis [BeT00]. Recent theoretical work on incremental gradient methods has focused, among others, on the aggregated gradient methods discussed in Section 2.1.5, on extensions to nondifferentiable and constrained problems, on combinations with the proximal algorithm, and on constrained versions where the constraints are treated by incremental projection (see Section 6.4 and the references cited there).

The incremental Newton and related methods have been considered by several authors in a stochastic approximation framework; e.g., by Sakrison [Sak66], Venter [Ven67], Fabian [Fab73], Poljak and Tsyp-

kin [PoT80], [PoT81], and more recently by Bottou and LeCun [BoL05], and Bhatnagar, Prasad, and Prashanth [BPP13]. Among others, these references quantify the convergence rate advantage that stochastic Newton methods have over stochastic gradient methods. Deterministic incremental Newton methods have received little attention (for a recent work see Gurbuzbalaban, Ozdaglar, and Parrilo [GOP14]). However, they admit an analysis that is similar to a deterministic analysis of the extended Kalman filter, the incremental version of the Gauss-Newton method (see Bertsekas [Ber96], and Moriyama, Yamashita, and Fukushima [MYF03]). There are also many stochastic analyses of the extended Kalman filter in the literature of estimation and control of dynamic systems.

Let us also note another approach to accelerate the theoretical convergence rate of incremental gradient methods, which involves using a larger than $O(1/k)$ stepsize and averaging the iterates (for analysis of the corresponding stochastic gradient methods, see Ruppert [Rup85], and Poljak and Juditsky [PoJ92], and for a textbook account, Kushner and Yin [KuY03]).

Section 2.2: The nonlinear programming textbooks cited earlier contain a lot of material on approximation methods. In particular, the literature on polyhedral approximation is extensive. It dates to the early days of nonlinear and convex programming, and it involves applications in data communication and transportation networks, and large-scale resource allocation. This literature will be reviewed in Chapter 4.

The research monographs by Fiacco and MacCormick [FiM68], and Bertsekas [Ber82a] focus on penalty and augmented Lagrangian methods, respectively. The latter book also contains a lot of material on smoothing methods and the proximal algorithm, including cases where nonquadratic regularization is involved, leading in turn to nonquadratic penalty terms in the augmented Lagrangian (e.g., logarithmic regularization and exponential penalty).

The proximal algorithm was proposed in the early 70s by Martinet [Mar70], [Mar72]. The literature on the algorithm and its extensions, spurred by the influential paper by Rockafellar [Roc76a], is voluminous, and reflects the central importance of proximal ideas in convex optimization and other problems. The ADMM, an important special case of the proximal context, was proposed by Glowinskii and Morocco [GIM75], and Gabay and Mercier [GaM76], and was further developed by Gabay [Gab79], [Gab83]. We refer to Section 5.4 for a detailed discussion of this algorithm, its applications, and its connections to more general operator splitting methods. Recent work involving proximal ideas has focused on combinations with other algorithms, such as gradient, subgradient, and coordinate descent methods. Some of these combined methods will be discussed in detail in Chapters 5 and 6.

E X E R C I S E S

2.1 (Convergence Rate of Steepest Descent and Gradient Projection for a Quadratic Cost Function)

Let f be the quadratic cost function,

$$f(x) = \tfrac{1}{2}x'Qx - b'x,$$

where Q is a symmetric positive definite matrix, and let m and M be the minimum and maximum eigenvalues of Q, respectively. Consider the minimization of f over a closed convex set X and the gradient projection mapping

$$G(x) = P_X\big(x - \alpha\nabla f(x)\big)$$

with constant stepsize $\alpha < 2/M$.

(a) Show that G is a contraction mapping and we have

$$\big\|G(x) - G(y)\big\| \leq \max\big\{|1-\alpha m|,\, |1-\alpha M|\big\}\|x-y\|, \qquad \forall\, x, y \in \Re^n,$$

and its unique fixed point is the unique minimum x^* of f over X. *Solution*: First note the nonexpansive property of the projection

$$\big\|P_X(x) - P_X(y)\big\| \leq \|x-y\|, \qquad \forall\, x, y \in \Re^n;$$

(use a Euclidean geometric argument, or see Section 3.2 for a proof). Use this property and the gradient formula $\nabla f(x) = Qx - b$ to write

$$\begin{aligned}\big\|G(x) - G(y)\big\| &= \big\|P_X\big(x - \alpha\nabla f(x)\big) - P_X\big(y - \alpha\nabla f(y)\big)\big\| \\ &\leq \big\|\big(x - \alpha\nabla f(x)\big) - \big(y - \alpha\nabla f(y)\big)\big\| \\ &= \big\|(I - \alpha Q)(x-y)\big\| \\ &\leq \max\big\{|1-\alpha m|, |1-\alpha M|\big\}\|x-y\|,\end{aligned}$$

where m and M are the minimum and maximum eigenvalues of Q. Clearly x^* is a fixed point of G if and only if $x^* = P_X\big(x^* - \alpha\nabla f(x^*)\big)$, which by the projection theorem, is true if and only if the necessary and sufficient condition for optimality $\nabla f(x^*)'(x - x^*) \geq 0$ for all $x \in X$ is satisfied. *Note*: In a generalization of this convergence rate estimate to the case of a nonquadratic strongly convex differentiable function f, the maximum eigenvalue M is replaced by the Lipschitz constant of ∇f and the minimum eigenvalue m is replaced by the modulus of strong convexity of f; see Section 6.1.

(b) Show that the value of α that minimizes the bound of part (a) is

$$\alpha^* = \frac{2}{M+m},$$

in which case

$$\big\|G(x) - G(y)\big\| \leq \left(\frac{M/m-1}{M/m+1}\right)\|x-y\|.$$

Note: The linear convergence rate estimate,

$$\|x_{k+1} - x^*\| \leq \left(\frac{M/m-1}{M/m+1}\right)\|x_k - x^*\|,$$

that this contraction property implies for steepest descent with constant stepsize is sharp, in the sense that there exist starting points x_0 for which the preceding inequality holds as an equation for all k (see [Ber99], Section 2.3).

2.2 (Descent Inequality)

This exercise deals with an inequality that is fundamental for the convergence analysis of gradient methods. Let X be a convex set, and let $f : \Re^n \mapsto \Re$ be a differentiable function such that for some constant $L > 0$, we have

$$\big\|\nabla f(x) - \nabla f(y)\big\| \leq L\|x-y\|, \qquad \forall\ x, y \in X.$$

Show that

$$f(y) \leq f(x) + \nabla f(x)'(y-x) + \frac{L}{2}\|y-x\|^2, \qquad \forall\ x, y \in X. \tag{2.70}$$

Proof: Let t be a scalar parameter and let $g(t) = f\big(x + t(y-x)\big)$. The chain rule yields $(dg/dt)(t) = \nabla f\big(x + t(y-x)\big)'(y-x)$. Thus, we have

$$\begin{aligned}
f(y) - f(x) &= g(1) - g(0) \\
&= \int_0^1 \frac{dg}{dt}(t)\,dt \\
&= \int_0^1 (y-x)'\nabla f\big(x + t(y-x)\big)\,dt \\
&\leq \int_0^1 (y-x)'\nabla f(x)\,dt + \left|\int_0^1 (y-x)'\big(\nabla f\big(x + t(y-x)\big) - \nabla f(x)\big)\,dt\right| \\
&\leq \int_0^1 (y-x)'\nabla f(x)\,dt + \int_0^1 \|y-x\| \cdot \|\nabla f\big(x + t(y-x)\big) - \nabla f(x)\|dt \\
&\leq (y-x)'\nabla f(x) + \|y-x\| \int_0^1 Lt\|y-x\|\,dt \\
&= (y-x)'\nabla f(x) + \frac{L}{2}\|y-x\|^2.
\end{aligned}$$

2.3 (Convergence of Steepest Descent with Constant Stepsize)

Let $f : \Re^n \mapsto \Re$ be a differentiable function such that for some constant $L > 0$, we have

$$\big\|\nabla f(x) - \nabla f(y)\big\| \leq L\|x - y\|, \qquad \forall\, x, y \in \Re^n. \tag{2.71}$$

Consider the sequence $\{x_k\}$ generated by the steepest descent iteration

$$x_{k+1} = x_k - \alpha \nabla f(x_k),$$

where $0 < \alpha < \frac{2}{L}$. Show that if $\{x_k\}$ has a limit point, then $\nabla f(x_k) \to 0$, and every limit point $\overline{x}$ of $\{x_k\}$ satisfies $\nabla f(\overline{x}) = 0$. *Proof*: We use the descent inequality (2.70) to show that the cost function is reduced at each iteration according to

$$\begin{aligned} f(x_{k+1}) &= f\big(x_k - \alpha \nabla f(x_k)\big) \\ &\leq f(x_k) + \nabla f(x_k)'\big(-\alpha \nabla f(x_k)\big) + \frac{\alpha^2 L}{2}\|\nabla f(x_k)\|^2 \\ &= f(x_k) - \alpha\left(1 - \frac{\alpha L}{2}\right)\|\nabla f(x_k)\|^2. \end{aligned}$$

Thus if there exists a limit point $\overline{x}$ of $\{x_k\}$, we have $f(x_k) \to f(\overline{x})$ and $\nabla f(x_k) \to 0$. This implies that $\nabla f(\overline{x}) = 0$, since $\nabla f(\cdot)$ is continuous by Eq. (2.71).

2.4 (Armijo/Backtracking Stepsize Rule)

Consider minimization of a continuously differentiable function $f : \Re^n \mapsto \Re$, using the iteration

$$x_{k+1} = x_k + \alpha_k d_k,$$

where d_k is a descent direction. Given fixed scalars β, and σ, with $0 < \beta < 1$, $0 < \sigma < 1$, and s_k with $\inf_{k \geq 0} s_k > 0$, the stepsize α_k is determined as follows: we set $\alpha_k = \beta^{m_k} s_k$, where m_k is the first nonnegative integer m for which

$$f(x_k) - f(x_k + \beta^m s_k d_k) \geq -\sigma \beta^m s_k \nabla f(x_k)' d_k.$$

Assume that there exist positive scalars c_1, c_2 such that for all k we have

$$c_1\big\|\nabla f(x_k)\big\|^2 \leq -\nabla f(x_k)' d_k, \qquad \|d_k\|^2 \leq c_2\big\|\nabla f(x_k)\big\|^2. \tag{2.72}$$

(a) Show that the stepsize α_k is well-defined, i.e., that it will be determined after a finite number of reductions if $\nabla f(x_k) \neq 0$. *Proof*: We have for all $s > 0$

$$f(x_k + s d_k) - f(x_k) = s \nabla f(x_k)' d_k + o(s).$$

Thus the test for acceptance of a stepsize $s > 0$ is written as

$$s\nabla f(x_k)'d_k + o(s) \leq \sigma s \nabla f(x_k)'d_k,$$

or using Eq. (2.72),

$$\frac{o(s)}{s} \leq (1-\sigma)c_1 \big\|\nabla f(x_k)\big\|^2,$$

which is satisfied for s in some interval $(0, \bar{s}_k]$. Thus the test will be passed for all m for which $\beta^m s_k \leq \bar{s}_k$.

(b) Show that every limit point $\overline{x}$ of the generated sequence $\{x_k\}$ satisfies $\nabla f(\overline{x}) = 0$. *Proof*: Assume, to arrive at a contradiction, that there is a subsequence $\{x_k\}_{\mathcal{K}}$ that converges to some $\overline{x}$ with $\nabla f(\overline{x}) \neq 0$. Since $\{f(x_k)\}$ is monotonically nonincreasing, $\{f(x_k)\}$ either converges to a finite value or diverges to $-\infty$. Since f is continuous, $f(\overline{x})$ is a limit point of $\{f(x_k)\}$, so it follows that the entire sequence $\{f(x_k)\}$ converges to $f(\overline{x})$. Hence,

$$f(x_k) - f(x_{k+1}) \to 0.$$

By the definition of the Armijo rule and the descent property $\nabla f(x_k)'d_k \leq 0$ of the direction d_k, we have

$$f(x_k) - f(x_{k+1}) \geq -\sigma\alpha_k \nabla f(x_k)'d_k \geq 0,$$

so by combining the preceding two relations,

$$\alpha_k \nabla f(x_k)'d_k \to 0. \tag{2.73}$$

From the left side of Eq. (2.72) and the hypothesis $\nabla f(\overline{x}) \neq 0$, it follows that

$$\limsup_{\substack{k\to\infty \\ k\in\mathcal{K}}} \nabla f(x_k)'d_k < 0, \tag{2.74}$$

which together with Eq. (2.73) implies that

$$\{\alpha_k\}_{\mathcal{K}} \to 0.$$

Since s_k, the initial trial value for α_k, is bounded away from 0, s_k will be reduced at least once for all $k \in \mathcal{K}$ that are greater than some iteration index $\overline{k}$. Thus we must have for all $k \in \mathcal{K}$ with $k > \overline{k}$,

$$f(x_k) - f\big(x_k + (\alpha_k/\beta)d_k\big) < -\sigma(\alpha_k/\beta)\nabla f(x_k)'d_k. \tag{2.75}$$

From the right side of Eq. (2.72), $\{d_k\}_{\mathcal{K}}$ is bounded, and it follows that there exists a subsequence $\{d_k\}_{\overline{\mathcal{K}}}$ of $\{d_k\}_{\mathcal{K}}$ such that

$$\{d_k\}_{\overline{\mathcal{K}}} \to \overline{d},$$

where $\overline{d}$ is some vector. From Eq. (2.75), we have

$$\frac{f(x_k) - f(x_k + \overline{\alpha}_k d_k)}{\overline{\alpha}_k} < -\sigma \nabla f(x_k)' d_k, \qquad \forall\ k \in \overline{\mathcal{K}},\ k \geq \overline{k},$$

where $\overline{\alpha}_k = \alpha_k/\beta$. By using the mean value theorem, this relation is written as

$$-\nabla f(x_k + \tilde{\alpha}_k d_k)' d_k < -\sigma \nabla f(x_k)' d_k, \qquad \forall\ k \in \overline{\mathcal{K}},\ k \geq \overline{k},$$

where $\tilde{\alpha}_k$ is a scalar in the interval $[0, \overline{\alpha}_k]$. Taking limits in the preceding relation we obtain

$$-\nabla f(\overline{x})' \overline{d} \leq -\sigma \nabla f(\overline{x})' \overline{d},$$

or

$$0 \leq (1 - \sigma) \nabla f(\overline{x})' \overline{d}.$$

Since $\sigma < 1$, it follows that

$$0 \leq \nabla f(\overline{x})' \overline{d},$$

a contradiction of Eq. (2.74).

2.5 (Convergence of Steepest Descent to a Single Limit [BGI95])

Let $f : \Re^n \mapsto \Re$ be a differentiable convex function, and assume that for some $L > 0$, we have

$$\big\|\nabla f(x) - \nabla f(y)\big\| \leq L \|x - y\|, \qquad \forall\ x, y \in \Re^n.$$

Let X^* be the set of minima of f, and assume that X^* is nonempty. Consider the steepest descent method

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k).$$

Show that $\{x_k\}$ converges to a minimizing point of f under each of the following two stepsize rule conditions:

(i) For some $\epsilon > 0$, we have

$$\epsilon \leq \alpha_k \leq \frac{2(1 - \epsilon)}{L}, \qquad \forall\ k.$$

(ii) $\alpha_k \to 0$ and $\sum_{k=0}^{\infty} \alpha_k = \infty$.

Notes: The original source [BGI95] also shows convergence to a single limit for a variant of the Armijo rule. This should be contrasted with a result of [Gon00], which shows that the steepest descent method with the exact line

minimization rule may produce a sequence with multiple limit points (all of which are of course optimal), even for a convex cost function. There is also a "local capture" theorem that applies to gradient methods for *nonconvex* continuously differentiable cost functions f and an isolated local minimum of f (a local minimum x^* that is unique within a neighborhood of x^*). Under mild conditions it asserts that there is an open sphere S_{x^*} centered at x^* such that once the generated sequence $\{x_k\}$ enters S_{x^*}, it converges to x^* (see [Ber82a], Prop. 1.12, or [Ber99], Prop. 1.2.5 and the references given there). *Abbreviated Proof*: Consider the stepsize rule (i). From the descent inequality (Exercise 2.2), we have for all k

$$f(x_{k+1}) \leq f(x_k) - \alpha_k \left(1 - \frac{\alpha_k L}{2}\right) \left\|\nabla f(x_k)\right\|^2 \leq f(x_k) - \epsilon^2 \left\|\nabla f(x_k)\right\|^2,$$

so $\left\{f(x_k)\right\}$ is monotonically nonincreasing and converges. Adding the preceding relation for all values of k and taking the limit as $k \to \infty$, we obtain for all $x^* \in X^*$,

$$f(x^*) \leq f(x_0) - \epsilon^2 \sum_{k=0}^{\infty} \left\|\nabla f(x_k)\right\|^2.$$

It follows that $\sum_{k=0}^{\infty} \left\|\nabla f(x_k)\right\|^2 < \infty$ and $\nabla f(x_k) \to 0$, and also

$$\sum_{k=0}^{\infty} \|x_{k+1} - x_k\|^2 < \infty, \tag{2.76}$$

since $\nabla f(x_k) = (x_k - x_{k+1})/\alpha_k$. Moreover any limit point of $\{x_k\}$ belongs to X^*, since $\nabla f(x_k) \to 0$ and f is convex.

Using the convexity of f, we have for all $x^* \in X^*$,

$$\begin{aligned} \|x_{k+1} - x^*\|^2 - \|x_k - x^*\|^2 - \|x_{k+1} - x_k\|^2 &= -2(x^* - x_k)'(x_{k+1} - x_k) \\ &= 2\alpha_k (x^* - x_k)' \nabla f(x_k) \\ &\leq 2\alpha_k \big(f(x^*) - f(x_k)\big) \\ &\leq 0, \end{aligned}$$

so that

$$\|x_{k+1} - x^*\|^2 \leq \|x_k - x^*\|^2 + \|x_{k+1} - x_k\|^2, \qquad \forall\ x^* \in X^*. \tag{2.77}$$

We now use Eqs. (2.76) and (2.77), and the Fejér Convergence Theorem (Prop. A.4.6 in Appendix A). From part (a) of that theorem it follows that $\{x_k\}$ is bounded, and hence it has a limit point $\overline{x}$, which must belong to X^* as shown earlier. Using this fact and part (b) of the theorem, it follows that $\{x_k\}$ converges to $\overline{x}$.

The proof for the case of the stepsize rule (ii) is similar. Using the assumptions $\alpha_k \to 0$ and $\sum_{k=0}^{\infty} \alpha_k = \infty$, and the descent inequality, we show that $\nabla f(x_k) \to 0$, that $\left\{f(x_k)\right\}$ converges, and that Eq. (2.76) holds. From this point, the preceding proof applies.

2.6 (Convergence of Gradient Method with Errors [BeT00])

Consider the problem of unconstrained minimization of a differentiable function $f : \Re^n \mapsto \Re$. Let $\{x_k\}$ be a sequence generated by the method

$$x_{k+1} = x_k - \alpha_k\big(\nabla f(x_k) + w_k\big),$$

where α_k is a positive stepsize, and w_k is an error vector satisfying for some positive scalars p and q,

$$\|w_k\| \leq \alpha_k\big(q + p\|\nabla f(x_k)\|\big), \qquad k = 0, 1, \ldots. \tag{2.78}$$

Assume that for some constant $L > 0$, we have

$$\big\|\nabla f(x) - \nabla f(y)\big\| \leq L\|x - y\|, \qquad \forall\ x, y \in \Re^n,$$

and that

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty. \tag{2.79}$$

Show that either $f(x_k) \to -\infty$ or else $f(x_k)$ converges to a finite value and $\lim_{k\to\infty} \nabla f(x_k) = 0$. Furthermore, every limit point $\overline{x}$ of $\{x_k\}$ satisfies $\nabla f(\overline{x}) = 0$. *Abbreviated Proof*: The descent inequality (2.70) yields

$$f(x_{k+1}) \leq f(x_k) - \alpha_k \nabla f(x_k)'\big(\nabla f(x_k) + w_k\big) + \frac{\alpha_k^2 L}{2}\big\|\nabla f(x_k) + w_k\big\|^2.$$

Using Eq. (2.78), we have

$$\begin{aligned} -\nabla f(x_k)'\big(\nabla f(x_k) + w_k\big) &\leq -\big\|\nabla f(x_k)\big\|^2 + \big\|\nabla f(x_k)\big\|\,\|w_k\| \\ &\leq -\big\|\nabla f(x_k)\big\|^2 + \alpha_k q\big\|\nabla f(x_k)\big\| + \alpha_k p\big\|\nabla f(x_k)\big\|^2, \end{aligned}$$

and

$$\begin{aligned} \frac{1}{2}\big\|\nabla f(x_k) + w_k\big\|^2 &\leq \big\|\nabla f(x_k)\big\|^2 + \|w_k\|^2 \\ &\leq \big\|\nabla f(x_k)\big\|^2 + \alpha_k^2\big(q^2 + 2pq\big\|\nabla f(x_k)\big\| + p^2\big\|\nabla f(x_k)\big\|^2\big). \end{aligned}$$

Combining the preceding three relations and collecting terms, it follows that

$$\begin{aligned} f(x_{k+1}) \leq f(x_k) &- \alpha_k(1 - \alpha_k L - \alpha_k p - \alpha_k^3 p^2 L)\big\|\nabla f(x_k)\big\|^2 \\ &+ \alpha_k^2(q + 2\alpha_k^2 pqL)\big\|\nabla f(x_k)\big\| + \alpha_k^4 q^2 L. \end{aligned}$$

Since $\alpha_k \to 0$, we have for some positive constants c and d, and all k sufficiently large

$$f(x_{k+1}) \leq f(x_k) - \alpha_k c\big\|\nabla f(x_k)\big\|^2 + \alpha_k^2 d\big\|\nabla f(x_k)\big\| + \alpha_k^4 q^2 L.$$

Using the inequality $\big\|\nabla f(x_k)\big\| \leq 1 + \big\|\nabla f(x_k)\big\|^2$, the above relation yields for all k sufficiently large

$$f(x_{k+1}) \leq f(x_k) - \alpha_k(c - \alpha_k d)\big\|\nabla f(x_k)\big\|^2 + \alpha_k^2 d + \alpha_k^4 q^2 L.$$

By applying the Supermartingale Convergence Theorem (Prop. A.4.4 in Appendix A), using also the assumption (2.79), it follows that either $f(x_k) \to -\infty$ or else $f(x_k)$ converges to a finite value and $\sum_{k=0}^{\infty} \alpha_k \big\|\nabla f(x_k)\big\|^2 < \infty$. In the latter case, in view of the assumption $\sum_{k=0}^{\infty} \alpha_k = \infty$, we must have $\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0$. This implies that $\nabla f(x_k) \to 0$; for a detailed proof of this last step see [BeT00]. This reference also provides a stochastic version of the result of this exercise. This result, however, requires a different line proof, which does not rely on supermartingale convergence arguments.

2.7 (Steepest Descent Direction for Nondifferentiable Cost Functions [BeM71])

Let $f : \Re^n \mapsto \Re$ be a convex function, and let us view the steepest descent direction at x as the solution of the problem

$$\begin{aligned} &\text{minimize} \quad f'(x;d) \\ &\text{subject to} \quad \|d\| \leq 1. \end{aligned} \tag{2.80}$$

Show that this direction is $-g^*$, where g^* is the vector of minimum norm in $\partial f(x)$. *Abbreviated Solution*: From Prop. 5.4.8 in Appendix B, $f'(x;\cdot)$ is the support function of the nonempty and compact subdifferential $\partial f(x)$, i.e.,

$$f'(x;d) = \max_{g\in\partial f(x)} d'g, \qquad \forall\ x, d \in \Re^n.$$

Since the sets $\big\{d \mid \|d\| \leq 1\big\}$ and $\partial f(x)$ are convex and compact, and the function $d'g$ is linear in each variable when the other variable is fixed, by the Saddle Point Theorem of Prop. 5.5.3 in Appendix B, it follows that

$$\min_{\|d\|\leq 1} \max_{g\in\partial f(x)} d'g = \max_{g\in\partial f(x)} \min_{\|d\|\leq 1} d'g,$$

and that a saddle point exists. For any saddle point (d^*, g^*), g^* maximizes the function $\min_{\|d\|\leq 1} d'g = -\|g\|$ over $\partial f(x)$, so g^* is the unique vector of minimum norm in $\partial f(x)$. Moreover, d^* minimizes $\max_{g\in\partial f(x)} d'g$ or equivalently $f'(x;d)$ [by Eq. (2.80)] subject to $\|d\| \leq 1$ (so it is a direction of steepest descent), and minimizes $d'g^*$ subject to $\|d\| \leq 1$, so it has the form

$$d^* = -\frac{g^*}{\|g^*\|}$$

[except if $0 \in \partial f(x)$, in which case $d^* = 0$].

2.8 (Two-Metric Projection Methods for Bound Constraints [Ber82a], [Ber82b])

Consider the minimization of a continuously differentiable function $f : \Re^n \mapsto \Re$ over the set

$$X = \big\{(x^1, \ldots, x^n) \mid \underline{b}^i \leq x^i \leq \bar{b}^i,\ i = 1, \ldots, n\big\},$$

where $\underline{b}^i$ and $\bar{b}^i$, $i = 1, \ldots, n$, are given scalars. The two-metric projection method for this problem has the form

$$x_{k+1} = P_X\big(x_k - \alpha_k D_k \nabla f(x_k)\big),$$

where D_k is a positive definite symmetric matrix.

(a) Construct an example of f and D_k, where x_k does not minimize f over X and $f(x_{k+1}) > f(x_k)$ for all $\alpha_k > 0$.

(b) For given $x_k \in X$, let $I_k = \{i \mid x_k^i = \underline{b}^i \text{ with } \partial f(x_k)/\partial x^i > 0 \text{ or } x_k^i = \bar{b}^i \text{ with } \partial f(x_k)/\partial x^i < 0\}$. Assume that D_k is diagonal with respect to I_k in the sense that $(D_k)_{ij} = (D_k)_{ji} = 0$ for all $i \in I_k$ and $j = 1, \ldots, n$, with $j \neq i$. Show that if x_k is not optimal, there exists $\bar{\alpha}_k > 0$ such that $f(x_{k+1}) < f(x_k)$ for all $\alpha_k \in (0, \bar{\alpha}_k]$.

(c) Assume that the nondiagonal portion of D_k is the inverse of the corresponding portion of $\nabla^2 f(x_k)$. Argue informally that the method can be reasonably expected to have superlinear convergence rate.

2.9 (Incremental Methods – Computational Exercise)

This exercise deals with the (perhaps approximate) solution of a system of linear inequalities $c_i' x \leq b_i$, $i = 1, \ldots, m$, where $c_i \in \Re^n$ and $b_i \in \Re$ are given.

(a) Consider a variation of the Kaczmarz algorithm that operates in cycles as follows. At the end of cycle k, we set $x_{k+1} = \psi_{m,k}$, where $\psi_{m,k}$ is obtained after the m steps

$$\psi_{i,k} = \psi_{i-1,k} - \frac{\alpha_k}{\|c_i\|^2} \max\big\{0,\ c_i' \psi_{i-1,k} - b_i\big\} c_i, \qquad i = 1, \ldots, m,$$

starting with $\psi_{0,k} = x_k$. Show that the algorithm can be viewed as an incremental gradient method for a suitable differentiable cost function.

(b) Implement the algorithm of (a) for two examples where $n = 2$ and $m = 100$. In the first example, the vectors c_i have the form $c_i = (\xi_i, \zeta_i)$, where ξ_i, ζ_i, as well as b_i, are chosen randomly and independently from $[-100, 100]$ according to a uniform distribution. In the second example, the vectors c_i have the form $c_i = (\xi_i, \zeta_i)$, where ξ_i, ζ_i are chosen randomly and independently within $[-10, 10]$ according to a uniform distribution, while b_i is chosen randomly and independently within $[0, 1000]$ according to a uniform distribution. Experiment with different starting points and stepsize choices, and deterministic and randomized orders of selection of the indexes i for iteration. Explain your experimental results in terms of the theoretical behavior described in Section 2.1.

2.10 (Convergence of the Incremental Gradient Method)

Consider the minimization of a cost function

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

where $f_i : \Re^n \mapsto \Re$ are continuously differentiable, and let $\{x_k\}$ be a sequence generated by the incremental gradient method. Assume that for some constants L, C, D, and all $i = 1, \ldots, m$, we have

$$\big\|\nabla f_i(x) - \nabla f_i(y)\big\| \leq L\|x - y\|, \qquad \forall\ x, y \in \Re^n,$$

and

$$\big\|\nabla f_i(x)\big\| \leq C + D\big\|\nabla f(x)\big\|, \qquad \forall\ x \in \Re^n.$$

Assume also that

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Show that either $f(x_k) \to -\infty$ or else $f(x_k)$ converges to a finite value and $\lim_{k\to\infty} \nabla f(x_k) = 0$. Furthermore, every limit point $\overline{x}$ of $\{x_k\}$ satisfies $\nabla f(\overline{x}) = 0$. *Abbreviated Solution*: The idea is to view the incremental gradient method as a gradient method with errors, so that the result of Exercise 2.6 can be used. For simplicity we assume that $m = 2$. The proof is similar when $m > 2$. We have

$$\psi_1 = x_k - \alpha_k \nabla f_1(x_k), \qquad x_{k+1} = \psi_1 - \alpha_k \nabla f_2(\psi_1).$$

By adding these two relations, we obtain

$$x_{k+1} = x_k + \alpha_k\big(-\nabla f(x_k) + w_k\big),$$

where

$$w_k = \nabla f_2(x_k) - \nabla f_2(\psi_1).$$

We have

$$\|w_k\| \leq L\|x_k - \psi_1\| = \alpha_k L\big\|\nabla f_1(x_k)\big\| \leq \alpha_k\big(LC + LD\big\|\nabla f(x_k)\big\|\big).$$

Thus Exercise 2.6 applies and the result follows.

2.11 (Convergence Rate of the Kaczmarz Algorithm with Random Projection [StV09])

Consider a consistent system of linear equations $c_i'x = b_i$, $i = 1, \ldots, m$, and assume for convenience that the vectors c_i have been scaled so that $\|c_i\| = 1$ for all i. A randomized version of the Kaczmarz method is given by

$$x_{k+1} = x_k - (c_{i_k}'x - b_{i_k})c_{i_k},$$

where i_k is an index randomly chosen from the set $\{1, \ldots, m\}$ with equal probabilities $1/m$, independently of previous choices. Let $P(x)$ denote the Euclidean projection of a vector $x \in \Re^n$ onto the set of solutions of the system, and let C be the matrix whose rows are $c_1, \ldots, c_m$. Show that

$$E\big\{\|x_{k+1} - P(x_{k+1})\|^2\big\} \leq \left(1 - \frac{\lambda_{min}}{m}\right) E\big\{\|x_k - P(x_k)\|^2\big\},$$

where λ_{min} is the minimum eigenvalue of the matrix $C'C$. *Hint:* Show that

$$\big\|x_{k+1} - P(x_{k+1})\big\|^2 \leq \big\|x_{k+1} - P(x_k)\big\|^2 = \big\|x_k - P(x_k)\big\|^2 - (c_{i_k}'x_{i_k} - b_{i_k})^2,$$

and take conditional expectation of both sides to show that

$$\begin{aligned} E\big\{\|x_{k+1} - P(x_{k+1})\|^2 \mid x_k\big\} &\leq \big\|x_k - P(x_k)\big\|^2 - \frac{1}{m}\|Cx_k - b\|^2 \\ &\leq \left(1 - \frac{\lambda_{min}}{m}\right)\big\|x_k - P(x_k)\big\|^2. \end{aligned}$$

2.12 (Limit Cycle of Incremental Gradient Method [Luo91])

Consider the scalar least squares problem

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}\big((b_1 - x)^2 + (b_2 - x)^2\big) \\ &\text{subject to} \quad x \in \Re, \end{aligned}$$

where b_1 and b_2 are given scalars, and the incremental gradient algorithm that generates x_{k+1} from x_k according to

$$x_{k+1} = \psi_k - \alpha(\psi_k - b_2),$$

where

$$\psi_k = x_k - \alpha(x_k - b_1),$$

and α is a positive stepsize. Assuming that $\alpha < 1$, show that $\{x_k\}$ and $\{\psi_k\}$ converge to limits $x(\alpha)$ and $\psi(\alpha)$, respectively. However, unless $b_1 = b_2$, $x(\alpha)$ and $\psi(\alpha)$ are neither equal to each other, nor equal to the least squares solution $x^* = (b_1 + b_2)/2$. Verify that

$$\lim_{\alpha \to 0} x(\alpha) = \lim_{\alpha \to 0} \psi(\alpha) = x^*.$$

2.13 (Convergence of Incremental Gradient Method for Linear Least Squares Problems)

Consider the linear least squares problem of minimizing

$$f(x) = \tfrac{1}{2} \sum_{i=1}^{m} \|z_i - C_i x\|^2$$

over $x \in \Re^n$, where the vectors z_i and the matrices C_i are given. Let x_k be the vector at the start of cycle k of the incremental gradient method that operates in cycles where components are selected according to a fixed order. Thus we have

$$x_{k+1} = x_k + \alpha_k \sum_{i=1}^{m} C_i'(z_i - C_i \psi_{i-1}),$$

where $\psi_0 = x_k$ and

$$\psi_i = \psi_{i-1} + \alpha_k C_i'(z_i - C_i \psi_{i-1}), \qquad i = 1, \ldots, m.$$

Assume that $\sum_{i=1}^{m} C_i' C_i$ is a positive definite matrix and let x^* be the optimal solution. Then:

(a) There exists $\overline{\alpha} > 0$ such that if α_k is equal to some constant $\alpha \in (0, \overline{\alpha}]$ for all k, $\{x_k\}$ converges to some vector $x(\alpha)$. Furthermore, the error $\|x_k - x(\alpha)\|$ converges to 0 linearly. In addition, we have $\lim_{\alpha \to 0} x(\alpha) = x^*$. *Hint:* Show that the mapping that produces x_{k+1} starting from x_k is a contraction mapping for α sufficiently small.

(b) If $\alpha_k > 0$ for all k, and

$$\alpha_k \to 0, \qquad \sum_{k=0}^{\infty} \alpha_k = \infty,$$

then $\{x_k\}$ converges to x^*. *Hint:* Use Prop. A.4.3 of Appendix A.

Note: The ideas of this exercise are due to [Luo91]. For a complete solution, see [BeT96], Section 3.2, or [Ber99], Section 1.5.

2.14 (Linear Convergence Rate of Incremental Gradient Method [Ber99], [NeB00])

This exercise quantifies the rate of convergence of the incremental gradient method to the "region of confusion" (cf. Fig. 2.1.11), for any order of processing the additive cost components, assuming these components are positive definite quadratic. Consider the incremental gradient method

$$x_{k+1} = x_k - \alpha \nabla f_k(x_k) \qquad k = 0, 1, \ldots,$$

where $f_0, f_1, \ldots,$ are quadratic functions with eigenvalues lying within some interval $[\gamma, \Gamma]$, where $\gamma > 0$. Suppose that for a given $\epsilon > 0$, there is a vector x^* such that

$$\left\| \nabla f_k(x^*) \right\| \leq \epsilon, \qquad \forall\ k = 0, 1, \ldots.$$

Show that for all α with $0 < \alpha \leq 2/(\gamma + \Gamma)$, the generated sequence $\{x_k\}$ converges to a $2\epsilon/\gamma$-neighborhood of x^*, i.e.,

$$\limsup_{k\to\infty} \|x_k - x^*\| \leq \frac{2\epsilon}{\gamma}.$$

Moreover the rate of convergence to this neighborhood is linear, in the sense that

$$\|x_k - x^*\| > \frac{2\epsilon}{\gamma} \qquad \Rightarrow \qquad \|x_{k+1} - x^*\| < \left(1 - \frac{\alpha\gamma}{2}\right) \|x_k - x^*\|,$$

while

$$\|x_k - x^*\| \leq \frac{2\epsilon}{\gamma} \qquad \Rightarrow \qquad \|x_{k+1} - x^*\| \leq \frac{2\epsilon}{\gamma}.$$

Hint: Let $f_k(x) = \frac{1}{2}x'Q_kx - b_k'x$, where Q_k is positive definite symmetric, and write

$$x_{k+1} - x^* = (I - \alpha Q_k)(x_k - x^*) - \alpha\nabla f_k(x^*).$$

For other related convergence rate results, see [NeB00] and [Sch14a].

2.15 (Proximal Gradient Method, ℓ_1-Regularization, and the Shrinkage Operation)

The proximal gradient iteration (2.27) is well suited for problems involving a nondifferentiable function component that is convenient for a proximal iteration. This exercise considers the important case of the ℓ_1 norm. Consider the problem

$$\begin{aligned} &\text{minimize} \quad f(x) + \gamma\|x\|_1 \\ &\text{subject to } x \in \Re^n, \end{aligned}$$

where $f : \Re^n \mapsto \Re$ is a differentiable convex function, $\|\cdot\|_1$ is the ℓ_1 norm, and $\gamma > 0$. The proximal gradient iteration is given by the gradient step

$$z_k = x_k - \alpha\nabla f(x_k),$$

followed by the proximal step

$$x_{k+1} \in \arg\min_{x\in\Re^n} \left\{\gamma\|x\|_1 + \frac{1}{2\alpha}\|x - z_k\|^2\right\};$$

[cf. Eq. (2.28)]. Show that the proximal step can be performed separately for each coordinate x^i of x, and is given by the so-called *shrinkage operation*:

$$x_{k+1}^i = \begin{cases} z_k^i - \alpha\gamma & \text{if } z_k^i > \alpha\gamma, \\ 0 & \text{if } |z_k^i| \leq \alpha\gamma, \\ z_k^i + \alpha\gamma & \text{if } z_k^i < -\alpha\gamma, \end{cases} \qquad i = 1, \ldots, n.$$

Note: Since the shrinkage operation tends to set many coordinates x_{k+1}^i to 0, it tends to produce "sparse" iterates.

2.16 (Determining Feasibility of Nonlinear Inequalities by Exponential Smoothing, [Ber82a], p. 314, [Sch82])

Consider the problem of finding a solution of a system of inequality constraints

$$g_i(x) \leq 0, \qquad i = 1, \ldots, m,$$

where $g_i : \Re^n \mapsto \Re$ are convex functions. A smoothing method based on the exponential penalty function is to minimize instead

$$f_{c,\lambda}(x) = \frac{1}{c} \ln \sum_{i=1}^{m} \lambda_i e^{cg_i(x)},$$

where $c > 0$ is some scalar, and the scalars λ_i, $i = 1, \ldots, m$, are such that

$$\lambda_i > 0, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} \lambda_i = 1.$$

(a) Show that if the system is feasible (or strictly feasible) the optimal value is nonpositive (or strictly negative, respectively). If the system is infeasible, then

$$\lim_{c \to \infty} \inf_{x \in \Re^n} f_{c,\lambda}(x) = \inf_{x \in \Re^n} \max \big\{ g_1(x), \ldots, g_m(x) \big\}.$$

(b) (*Computational Exercise*) Apply the incremental gradient method and the incremental Newton method for minimizing $\sum_{i=1}^{m} e^{cg_i(x)}$ [which is equivalent to minimizing $f_{c,\lambda}(x)$ with $\lambda_i \equiv 1/m$], for the case

$$g_i(x) = c_i' x - b_i, \qquad i = 1, \ldots, m,$$

where (c_i, b_i) are randomly generated as in the two problems of Exercise 2.9(b). Experiment with different starting points and stepsize choices, and deterministic and randomized orders of selection of the indexes i for iteration.

(c) Repeat part (b) where the problem is instead to minimize $f(x) = \max_{i=1,\ldots,m} g_i(x)$ and the exponential smoothing method of Section 2.2.5 is used, possibly with the augmented Lagrangian update (2.67) of λ. Compare three methods of operation: (1) c is kept constant and λ is updated, (2) c is increased to ∞ and λ is kept constant, and (3) c is increased to ∞ and λ is updated.