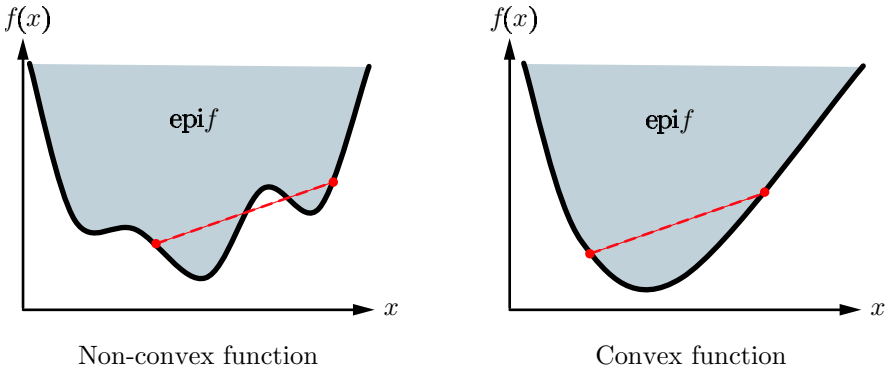

Non-convex function Convex function

Figure 5.1. One-dimensional example of non-convex (left) and a convex (right) functions. The shaded area represents the epigraph of the function. The dashed line shows a chord connecting two points on the graph of the function.

In the one-dimensional case (see Figure 5.1), this definition can be easily visualized geometrically: the graph of a convex function is always below the chord connecting any two points on it. This property can be formulated as $f(x) \geq f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0)$, for all $x, x_0 \in \mathbb{X}$. Obviously, the definition of a convex function is very similar to the definition of a convex set. A convex function can be alternatively defined as a function whose *epigraph* (set of points lying on or above the graph of the function) is a convex set, which generalizes the one-dimensional intuition (see Problem 5.6).

From this geometric interpretation, it is clear why a convex function cannot have more than one local minimum: indeed, if we had two local minima, going from one to another would require the function graph to first go up and then once more down, thus violating the convexity property. Consequently, for a convex function, a local minimum is necessarily the global one[1] (a rigorous proof is left as Problem 5.8). This property makes convex functions especially favorable for optimization: using local optimization methods, we can guarantee reaching the global minimum.

5.3 Unconstrained optimization algorithms

The traveler from our example, now equipped with the optimality conditions, knows where to stop, yet now he has to figure out how to move in order to get to the minimum. To make the situation even more dramatic, imagine that our traveler has no topographic map of the area and the visibility conditions are poor. Having no possibility to plan the entire route from his current location to the bottom of the valley (the local minimum), the traveler has to make small steps that lead him to lower altitude, based only on local information. For example, he can make a step in the direction in which the slope of the

input : objective function f.
output : minimizer x^* of $f(x)$.
initialization: some initial $x^{(0)}$ and $k = 0$.

1 **repeat**
2 Determine a *descent direction* $d^{(k)}$.
3 Choose a *step size* $\alpha^{(k)}$.
4 Update $x^{(k+1)} \longleftarrow x^{(k)} + \alpha^{(k)} d^{(k)}$.
5 $k \longleftarrow k + 1$.
6 **until** *convergence*
7 $x^* \approx x^{(k)}$.

Algorithm 5.1. Generic minimization algorithm.

mountain inclines downward. Then, he finds a "downward" direction in the new point and makes another step and so on until the destination is reached. Unconstrained minimization algorithms follow essentially the same idea, which can be formalized as the following iterative procedure:

The algorithm starts with some *initialization*, $x^{(0)}$, which can be derived from some *a priori* information about the optimal solution or a random vector. It then chooses a vector $d^{(0)}$ and a scalar $\alpha^{(0)}$ such that $f(x^{(0)}) > f(x^{(0)} + \alpha^{(0)} d^{(0)})$; d is called a *descent direction* and α a *step size*. The current point $x^{(0)}$ is replaced with $x^{(1)} = x^{(0)} + \alpha^{(0)} d^{(0)}$ and the process is repeated iteratively. An optimization algorithm is said to *converge* if it produces a convergent sequence of points $x^{(0)}, x^{(1)}, \ldots \to x^*$ (called a *minimizing sequence*), such that x^* is a local minimizer of the objective function f. When f is convex, x^* is also its global minimizer; otherwise we say that the algorithm is *locally convergent*.

Optimization algorithms differ in three basic components: Step 3 (the choice of the descent direction), Step 4 (the choice of the step size), and Step 5 (the *stopping criterion*). We do not include Step 1 (initialization) into this list, because it is problem- rather than algorithm-specific. An ideal way to stop the optimization is when $|f(x^{(k)}) - f(x^*)| = 0$ or $\|x^{(k)} - x^*\| = 0$. However, because we do not know x^* in advance, this criterion is unusable. The first-order optimality condition tells us that $\nabla f = 0$ in the minimum, and this condition can be used to stop the algorithm. In practice, due to the use of finite-precision arithmetic, it is unlikely that the gradient will vanish completely. For this reason, practical optimization algorithms are usually stopped when $\|\nabla f\| \leq \epsilon_{\mathrm{g}}$, for some tolerance ϵ_{g}. It is also common to stop the algorithm when the relative change of the function value $(f(x^{(k)}) - f(x^{(k+1)}))/f(x^{(k)})$ drops below some tolerance threshold ϵ_f, or when the step size $\|x^{(k+1)} - x^{(k)}\|$ becomes sufficiently small.[2] Combinations of one or more of these stopping criteria are often used.

Provided that d is a descent direction, it is guaranteed that a sufficiently small step in this direction will decrease the value of f. That is, if we define the

one-dimensional function $f_d(\alpha) = f(x + \alpha d)$, it is guaranteed that $f_d(\alpha) < f_d(0)$ for a sufficiently small α. Step size selection determines where along the ray $\{x + \alpha d : \alpha \geq 0\}$ the next iterate will be. The simplest choice is the *constant step size*, $\alpha^{(k)} = \alpha_0$. Though widely used, such a strategy is problematic. Indeed, d is a decrease direction only in the proximity of x; too large a step size may increase the function resulting in oscillatory behavior of $f(x^{(k)})$ and often preventing convergence. A possible remedy is reducing the step size, however, in this case a minimization algorithm will suffer from slow convergence. For this reason, a far better strategy is to chose $\alpha^{(k)}$ adaptively by searching for suitable values along the ray $\{x + \alpha d : \alpha \geq 0\}$. Such a procedure is usually referred to as *line search*.[3]

The best way of choosing the step size is by using *exact line search*, in which α is chosen to minimize f along the ray $\{x + \alpha d : \alpha \geq 0\}$:

$$\alpha = \arg\min_{\alpha \geq 0} f(x + \alpha d). \tag{5.3}$$

There exists a variety of numerical procedures for solving the above one-dimensional minimization problem [30, 46]. In some special cases, the solution may have an analytic form. Exact line search is used when its complexity is significantly lower than the complexity of computing d itself.

Many times, the exact minimizer of $f(x + \alpha d)$ is not necessary or too expensive to find. Line search methods that find a step size that reduces f "sufficiently" are called *inexact*. One of the most popular versions of such line searches is known as the *Armijo rule* or *backtracking line search* (Algorithm 5.2).

Backtracking line search starts with some initial step size α_0 and then gradually reduces it by the factor β until the condition $f(x + \alpha d) \leq f(x) + \sigma\alpha\nabla f(x)^{\mathrm{T}} d$ is satisfied. The geometric meaning of this condition is visualized in Figure 5.2: on one hand, for a sufficiently small α, the one-dimensional function $f(x + \alpha d)$ behaves very similarly to its first-order Taylor approximation $f(x) + \alpha\nabla f(x)^{\mathrm{T}} d$, hence f can be decreased by $\alpha\nabla f(x)^{\mathrm{T}} d$. On the other hand, for larger values of the step size that are desired for faster convergence, $f(x + \alpha d)$ may lie above the line $f(x) + \alpha\nabla f(x)^{\mathrm{T}} d$ and sometimes a too large

input : descent direction d, objective function $f(x)$ and its gradient $\nabla f(x)$, parameters $\sigma \in (0, 0.5)$ and $\beta \in (0, 1)$.
output : step size α.
initialization: initial step size α_0.

1 $\alpha \longleftarrow \alpha_0$
2 **while** $f(x + \alpha d) > f(x) + \sigma\alpha\nabla f(x)^{\mathrm{T}} d$ **do**
3 $\quad \alpha \longleftarrow \beta\alpha$.
4 **end**

Algorithm 5.2. Backtracking line search (Armijo rule).

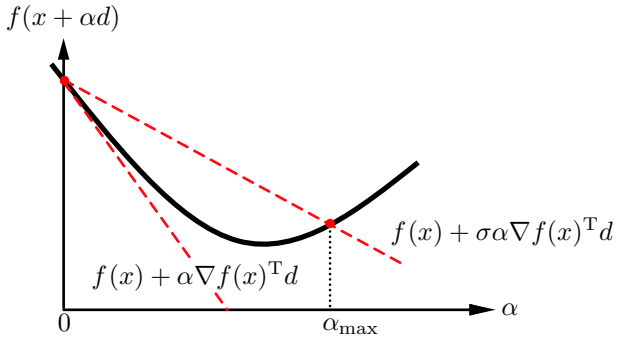

Figure 5.2. Visualization of the Armijo rule applied to a one-dimensional function $f(x+\alpha d)$. The lower dashed line shows the linear extrapolation of f, whereas the upper dashed line has a slope smaller by a factor of σ. Armijo rule accepts any value of α, for which the function lies between the two lines, i.e., $0 \leq \alpha \leq \alpha_{\max}$.

α may even increase the value of the function. The Armijo rule provides a reasonable trade-off between the two situations by accepting a decrease of factor σ of that suggested by first-order extrapolation. Typically, σ ranges between 0.01 to 0.3 and β between 0.1 (fast, yet crude search) and 0.8 (more accurate, yet slower search).

Both exact and inexact types of line search guarantee a decrease of f at every iteration of a minimization algorithm, which produces a monotonically non-increasing sequence $f(x^{(0)}) \geq f(x^{(1)}) \geq \cdots \geq f(x^{(k)}) \geq \cdots$ of function values. For this reason, algorithms that use line search are often termed *safeguarded*.

5.4 The quest for a descent direction

Thus far, we have assumed that the decrease direction d was given. We will now explore several ways to find it. Observe the first-order Taylor approximation of $f(x+d)$ around x,

$$f(x+d) \approx f(x) + \nabla f(x)^{\mathrm{T}} d.$$

The term $\nabla f(x)^{\mathrm{T}} d$ is called the *directional derivative* of f at x in the direction d. It describes the approximate change in f for a small step in the direction d. Observe that $f(x+d) < f(x)$ if the directional derivative $\nabla f(x)^{\mathrm{T}} d$ is negative. In other words, a descent direction must form an acute angle with the negative gradient.

Intuitively, to make the descent the steepest, it is desirable to make $\nabla f(x)^{\mathrm{T}} d$ as negative as possible. Because the term $\nabla f(x)^{\mathrm{T}} d$ is linear, it is unbounded below and thus we can make it as negative as we like by selecting