4.2 Proximal gradient method

Consider the problem

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

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

The *proximal gradient method* is

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

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

When ∇f is Lipschitz continuous with constant L, this method can be shown to converge with rate $O(1/k)$ when a fixed step size $\lambda^k = \lambda \in (0, 1/L]$ is used. (As discussed in [61], the method will actually converge for step sizes smaller than $2/L$, not just $1/L$, though for step sizes larger than $1/L$, the method is no longer a 'majorization-minimization method' as discussed in the next section.) If L is not known, the step sizes λ^k can be found by a line search [18, §2.4.3]; that is, their values are chosen in each step.

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

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

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

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

4.2.1 Interpretations

The first two interpretations given below are due to Beck and Teboulle [18]; we have repeated their discussion here for completeness. In the context of image processing problems, the majorization-minimization interpretation appeared in some even earlier papers by Figueiredo et al. [85, 83]. We also mention that in some special cases, additional interpretations are possible; for example, applying the method to the lasso can be interpreted as a kind of EM algorithm [84].

Majorization-minimization. We first interpret the proximal gradient method as an example of a *majorization-minimization algorithm*, a large class of algorithms that includes the gradient method, Newton's method, and the EM algorithm as special cases; see, *e.g.*, [106].

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

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

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

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

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

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

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

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

It then follows that the function q_λ given by

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

assuming here that g is also differentiable.

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

For example, we can consider the discretization

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

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

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

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

4.3 Accelerated proximal gradient method

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

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

where $\omega^k \in [0,1)$ is an extrapolation parameter and λ^k is the usual step size. (We let $\omega^0 = 0$, so the value x^{-1} appearing in the first extra step doesn't matter.) These parameters must be chosen in specific ways to achieve the convergence acceleration. One simple choice [192] takes

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

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

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

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