10-725/36-725: Convex Optimization **Spring 2015**

Lecture 8: February 9

Lecturer: Ryan Tibshirani *Scribes: Kartikeya Bhardwaj, Sangwon Hyun, Irina Cazan*

8.1 Proximal Gradient Descent

In the previous lecture, we learned about subgradient method in which we choose an initial $x^{(0)} \in \mathbb{R}^n$ and then repeat for a convex f: $x^{(k)} = x^{(k-1)} - t_k g^{(k-1)}, \quad k = 1, 2, 3, \ldots$; where $g^{(k-1)} \in \partial f(x^{(k-1)})$.

We also learned about the convergence rate of usual (batch) and stochastic subgradient methods. In particular, batch method has a convergence rate of $\mathcal{O}(G^2_{batch}/\epsilon^2)$, while cyclic and randomized methods have convergence rates of $\mathcal{O}(m(mG)^2/\epsilon^2)$ and $\mathcal{O}(mG^2/\epsilon^2)$ respectively. This implies that for non-smooth functions subgradient method achieves a lower bound of $\mathcal{O}(\frac{1}{\epsilon^2})$ which is very slow. Now consider a function f that can be decomposed into two functions as follows:

$$f(x) = g(x) + h(x) \tag{8.1}$$

where, g is convex, differentiable and $dom(g) = \mathbb{R}^n$. Also, h is convex and not necessarily differentiable, but is simple. In this lecture, we will learn that for a function f satisfying above conditions, we can recover a convergence rate of $\mathcal{O}(\frac{1}{\epsilon})$.

8.1.1 Decomposable functions

For a differentiable function f, recall that the gradient update $x^+ = x - t \cdot \nabla f(x)$ is derived using quadratic approximation (replace $\nabla^2 f(x)$ by $\frac{1}{t} I$):

$$x^+ = \underset{z}{\mathrm{argmin}} \quad \underbrace{f(x) + \nabla f(x)^T (z - x) + \frac{1}{2t}\|z - x\|_2^2}_{\widetilde{f}_t(z)} \tag{8.2}$$

For a decomposable function $f(x) = g(x) + h(x)$ described above (Equation 8.1), let us use this quadratic approximation for the differentiable function g. Then,

$$\begin{aligned} x^+ &= \underset{z}{\mathrm{argmin}} \quad \widetilde{g}_t(z) + h(z) \\ &= \underset{z}{\mathrm{argmin}} \quad g(x) + \nabla g(x)^T (z - x) + \frac{1}{2t}\|z - x\|_2^2 + h(z) \\ &= \underset{z}{\mathrm{argmin}} \quad \frac{1}{2t}\|z - (x - t\nabla g(x))\|_2^2 + h(z) \end{aligned} \tag{8.3}$$

The first term here signifies staying close to the gradient update for g while at the same time, making the value of h small using the second term.

8.1.2 Proximal Mapping and Proximal Gradient Descent

Next, we define proximal mapping as a function of h and t as follows:

$$prox_{h,t}(x) = \underset{z}{\text{argmin}} \ \ \frac{1}{2t}\|z-x\|_2^2 + h(z) \tag{8.4}$$

Now, using Equation 8.4, Equation 8.3 can be written as:

$$\begin{aligned} x^+ &= \underset{z}{\text{argmin}} \ \ \frac{1}{2t}\|z-(x-t\nabla g(x))\|_2^2 + h(z) \\ &= prox_{h,t}(x-t\nabla g(x)) \end{aligned} \tag{8.5}$$

For simplicity, from here onwards we will write proximal map as a function of t alone. Therefore, proximal gradient descent can be defined as follows– Choose initial $x^{(0)}$ and then repeat:

$$x^{(k)} = \ prox_{t_k}(x^{(k-1)} - t_k \nabla g(x^{(k-1)})), \quad k = 1, 2, 3, \ldots \tag{8.6}$$

This can be further expressed in a more familiar form:

$$x^{(k)} = x^{(k-1)} - t_k \cdot G_{t_k}(x^{(k-1)}) \tag{8.7}$$

where G_t is a generalized gradient of f,

$$G_t(x) = \frac{x - prox_t(x - t\nabla g(x))}{t} \tag{8.8}$$

Key Points:

- Proximal mapping $prox_t(\cdot)$ can be computed analytically for a lot of important h functions
- The mapping $prox_t(\cdot)$ doesn't depend on g at all, only on h
- This also means that g can be a complicated function; all we need to do is to compute its gradient
- Each iteration of proximal gradient descent evaluates $prox_t(\cdot)$ once which can be cheap or expensive depending on h

8.1.3 Iterative soft-thresholding algorithm (ISTA)

For a given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, the lasso criterion is given by:

$$f(\beta) = \frac{1}{2}\underbrace{\|y - X\beta\|_2^2}_{g(\beta)} + \underbrace{\lambda\|\beta\|_1}_{h(\beta)} \tag{8.9}$$

The proximal mapping for the lasso objective is computed as follows:

$$\begin{aligned} prox_t(\beta) &= \underset{z \in \mathbb{R}^n 0}{\text{argmin}} \frac{1}{2t}\|\beta - z\|_2^2 + \lambda\|z\|_1 \\ &\equiv \underset{z \in \mathbb{R}^n 0}{\text{argmin}} \frac{1}{2}\|\beta - z\|_2^2 + \lambda t\|z\|_1 \\ &= S_{\lambda t}(\beta) \end{aligned} \tag{8.10}$$

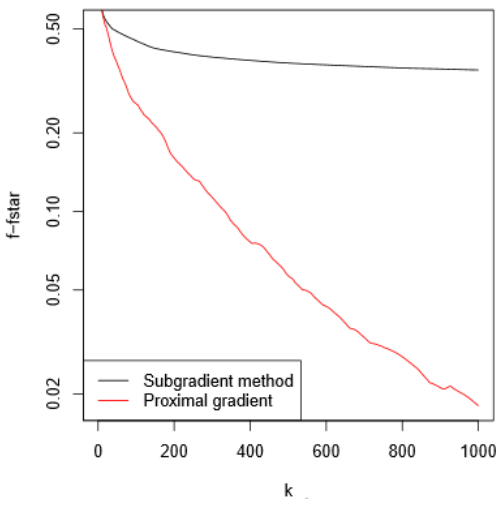

Figure 8.1: Example of proximal gradient descent (ISTA) vs. subgradient method convergence rates

where, from last lecture, we know that $S_\lambda(\beta)$ is the soft-thresholding operator given by:

$$[S_\lambda(\beta)]_i = \begin{cases} \beta_i - \lambda & \text{if } \beta_i > \lambda \\ 0 & \text{if } -\lambda \leq \beta_i \leq \lambda \\ \beta_i + \lambda & \text{if } \beta_i < -\lambda \end{cases} \tag{8.11}$$

Therefore, the proximal map for lasso objective is calculated by soft-thresholding β by amount λt. Next, we use the gradient of g, $\nabla g(\beta)$ which is same as the gradient of least squares function, *i.e.* $\nabla g(\beta) = -X^T(y - X\beta)$. From this, we obtain the following proximal gradient update (using Equations 8.5, 8.10, 8.11):

$$\beta^+ = S_{\lambda t}(\beta + t \cdot X^T(y - X\beta)) \tag{8.12}$$

This simple algorithm calculates the lasso solution and is known as Iterative soft-thresholding algorithm (ISTA). Fig. 8.1 shows the comparison between subgradient method and the proximal gradient descent (ISTA) for computing lasso solutions. Clearly, the proximal gradient descent approach is much faster.

8.1.4 Convergence Analysis

For a criterion $f(x) = g(x) + h(x)$, we make the following assumptions:

- The function g is convex, differentiable, $dom(g) = \mathbb{R}^n$, and ∇g is Lipschitz continuous with $L > 0$
- The function h is convex and its proximal map (Equation 8.4) can be evaluated

Then, we have the following Theorem:

Theorem 8.1 *Proximal gradient descent with fixed step size $t \leq 1/L$ satisfies*

$$f(x^{(k)}) - f* \leq \frac{\|x^{(0)} - x*\|_2^2}{2tk} \tag{8.13}$$

This implies that the proximal gradient descent has a convergence rate of $\mathcal{O}(1/k)$ or $\mathcal{O}(1/\epsilon)$.

Proximal gradient descent up till convergence analysis has already been scribed.

8.1.5 Backtracking Line Search

Backtracking line search for proximal gradient descent is similar to gradient descent but operates on g, the smooth part of f. First fix a parameter $0 < \beta < 1$, and at each iteration, start with $t = 1$, and while

$$g(x - tG_t(x)) > g(x) - t\nabla g(x)^T G_t(x) + \frac{t}{2}\|G_t(x)\|_2^2 \tag{8.14}$$

shrink $t = \beta t$. Else, perform proximal gradient update. Under the same assumptions we used in the original proximal gradient method's convergence analysis:

- The function g is convex, differentiable, $dom(g) = \mathbb{R}^n$, and ∇g is Lipschitz continuous with $L > 0$
- The function h is convex and its proximal map (Equation 8.4) can be evaluated

we get the same convergence rate as before:

$$f(x^{(k)}) - f* \leq \frac{\|x^{(0)} - x*\|_2^2}{2t_{\min}k} \tag{8.15}$$

where $t_{\min} = \min\{1, \beta/L\}$

8.2 Examples

8.2.1 Matrix Completion

This was famously used in the fairly recent Netflix Prize competition. Given a matrix $Y \in \mathbb{R}^{m\times n}$ where you only observe the entries $Y_{ij}, (i,j) \in \Omega$. Suppose we want to fill in missing entries (e.g. for a recommender system), so we solve a *matrix completion* problem:

$$\min_{B\in\mathbb{R}^{m\times n}} \frac{1}{2}\sum_{(i,j)\in\Omega}(Y_{ij} - B_{ij})^2 + \lambda\|B\|_{\text{tr}} \tag{8.16}$$

where $\|B\|_{\text{tr}}$ is the trace (nuclear) norm of B:

$$\|B\|_{\text{tr}} = \sum_{i=1}^{r}\sigma_i(B) = \text{trace}(\sqrt{B^T B}) \tag{8.17}$$

and $r = \text{rank}(B)$ and $\sigma_1(B) \geq \cdots \geq \sigma_r(B) \geq 0$ are singular values of matrix X. In short, this is a trace-norm-regularized entry-wise square-loss minimization problem. The trace norm is a convex relaxation of the rank equality constraint!

Now, define $P_\Omega(B) = \begin{cases} B_{ij} & (i,j) \in \Omega \\ 0 & (i,j) \in \Omega \end{cases}$, which is the projection operator onto an observed index set Ω. Then, the criterion is:

$$f(B) = \frac{1}{2}\|P_\Omega(Y) - P_\Omega(B)\|_F^2 + \lambda\|B\|_{\text{tr}} \tag{8.18}$$

the sum of $g(B)$ and $h(B)$; this is just a matrix representation of 8.16! Now, the two ingredients needed for proximal gradient descent are the gradient of the smooth part g and the prox function:

- Gradient: $\nabla g(B) = -(P_\Omega(Y) - P_\Omega(B))$

- Prox Function:

$$\text{prox}_t(B) = \underset{Z \in \mathbb{R}^{m \times n}}{\text{argmin}} \frac{1}{2t}\|B - Z\|_F^2 + \lambda \|Z\|_{\text{tr}} \tag{8.19}$$

We claim that $\text{prox}_t(B) = S_{\lambda t}(B)$, the matrix soft-thresholding at the level λt, where

$$S_\lambda(B) = U\Sigma_\lambda V^T \tag{8.20}$$

for the SVD $B = U\Sigma V^T$ and diagonal Σ_λ such that

$$(\Sigma_\lambda)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\} \tag{8.21}$$

Subgradient optimality tells us that the optimal solution to 8.18 should satisfy, for some

$$0 \in Z - B + \lambda t \partial \|Z\|_{tr} \tag{8.22}$$

So, let's set $Z = S_{\lambda t}(B)$ and see that this indeed 8.22. Denoting the SVD of B and $B = U\Sigma V^T$, we can see that the soft-thresholded version Z has a singular value decomposition of $U\Sigma_{\lambda t}V^T$ where $\Sigma_{\lambda t}$ has diagonal elements thresholded above at λt (this is directly from the definition of the soft-thresholding operator $S_{\lambda t}$). Plugging the SVD into 8.22, we need only verify the following:

$$0 \in U\Sigma_{\lambda t}V^T - U\Sigma V^T + \lambda t \partial \|U\Sigma_{\lambda t}V^T\| \tag{8.23}$$

$$\iff \frac{1}{\lambda t}\left[U\Sigma V^T - U\Sigma_{\lambda t}V^T\right] \in \partial \|U\Sigma_{\lambda t}V^T\| \tag{8.24}$$

Take the left hand side and rearrange to make

$$\frac{1}{\lambda t}U\Sigma V^T - U\Sigma_{\lambda t}V^T = UV^T + \frac{1}{\lambda t}U(\frac{\Sigma - \Sigma_{\lambda t}}{\lambda t} - I)V^T = UV^T + W \tag{8.25}$$

Because the soft thresholding operation on $B = U\Sigma V^T$ gives:

$$\Sigma_{\lambda t} = \text{diag}\left((\Sigma_{11} - \lambda t)^+, (\Sigma_{22} - \lambda t)^+, \cdots\right) \tag{8.26}$$

we can verify the following

$$\frac{1}{\lambda t}(\Sigma - \Sigma_{\lambda t}) - I = \begin{pmatrix} \frac{1}{\lambda t}[\Sigma_{11} - (\Sigma_{11} - \lambda t)^+] & \cdots & \cdots & 0 \\ \vdots & \frac{1}{\lambda t}[\Sigma_{22} - (\Sigma_{22} - \lambda)^+] & \vdots & 0 \\ \vdots & \cdots & \ddots & 0 \\ 0 & 0 & 0 & \ddots \end{pmatrix} - I \tag{8.27}$$

$$= \begin{pmatrix} \frac{1}{\lambda t}\min[\lambda t, \Sigma_{11}] & \cdots & \cdots & 0 \\ \vdots & \frac{1}{\lambda t}\min[\lambda t, \Sigma_{22}] & \vdots & 0 \\ \vdots & \cdots & \ddots & 0 \\ 0 & 0 & 0 & \ddots \end{pmatrix} - I \tag{8.28}$$

$$= \begin{pmatrix} \min[1, \frac{\Sigma_{11}}{\lambda t}] - 1 & \cdots & \cdots & \\ \vdots & \min[1, \frac{\Sigma_{22}}{\lambda t}] - 1 & \vdots & 0 \\ \vdots & \cdots & \ddots & 0 \\ 0 & 0 & 0 & \ddots \end{pmatrix} \tag{8.29}$$

whose diagonal entries are all smaller than 0 so that $\|W\|_{\text{op}} \leq 0$. It is easy to see that $U^T W = WV = 0$. Therefore, the left hand side of 8.24 is indeed a subgradient of the trace norm of Z (right hand side of 8.24):

$$\frac{1}{\lambda t}\left[U\Sigma V^T - U\Sigma_{\lambda t} V^T\right] = U^T V + W \in \partial\|U\Sigma_{\lambda t} V^T\| \tag{8.30}$$

$$= \{UV^T + W_0 : \|W\|_{\text{op}} \leq 1, U^T W_0 = 0, W_0 V = 0\} \tag{8.31}$$

where the proof of the last equality can be found in this paper.

Using this, we can see that the proximal gradient update step is:

$$B^+ = S_{\lambda t}\left(B + t(P_\Omega(Y) - P_\Omega(B))\right) \tag{8.32}$$

We can also see that $\nabla g(B)$ is Lipschitz continuous with $L = 1$, so we can choose fixed step size $t = 1$ for a simpler fixed-step proximal update with proved rate $O(1/\epsilon)$. The update is now:

$$B^+ = S_{\lambda t}\left(t(P_\Omega(Y) - P_\Omega^\perp(B))\right) \tag{8.33}$$

where $P^\perp(\cdot)$ projects onto the unobserved set. This is the *soft-impute* algorithm, a simple and effective method for matrix completion.

8.3 Special cases

Proximal gradient descent is also called composite gradient descent, or generalized gradient descent. The latter is because we can see that special cases of proximal gradient descent give some familiar/interesting forms; when minimizing $f = g + h$

- $h = 0$ gives gradient descent
- $h = I_C$ gives projected gradient descent
- $g = 0$ gives proximal minimzation algorithm.

All of these have $O(1/\epsilon)$ convergence rate. Let's examine projected gradient descent.

8.3.1 Projected gradient Descent

Given a closed, convex set $C \in \mathbb{R}^n$, the following two problems are equivalent:

$$\min_{x \in C} g(x) \Longleftrightarrow \min_{x \in \mathbb{R}^n} g(x) + I_C(x) \tag{8.34}$$

where $I_C(x) = \begin{cases} 0 & x \in C \\ \infty & x \notin C \end{cases}$ is the indicator function of C. The proximal operator is thus

$$\text{prox}_t(x) = \underset{z \in \mathbb{R}^n}{\text{argmin}} \frac{1}{2t}\|x - z\|_2^2 + I_C(z) \tag{8.35}$$

$$= \underset{z \in C}{\text{argmin}} \|x - z\|_2^2 = P_C(x) \tag{8.36}$$

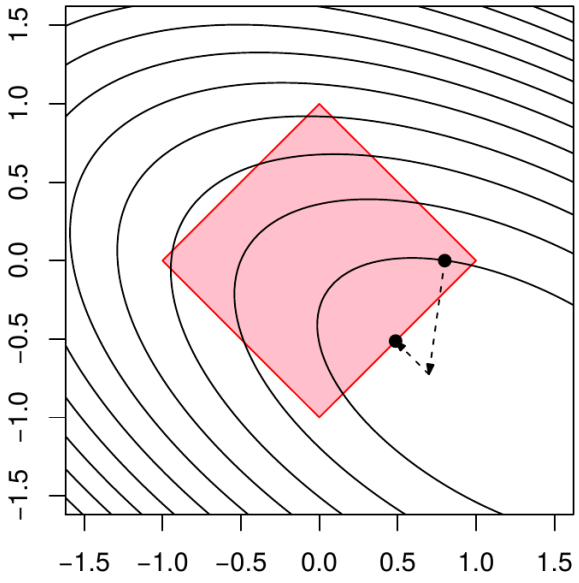

Figure 8.1: Projected gradient descent.

is the projection operator onto set C, so that the proximal gradient update step is to, at each step, project the usual gradient update back into the set C. (see figure 8.1) This *projected gradient descent* update is:

$$x^+ = P_C(x - t\nabla g(x)) \tag{8.37}$$

We can also do projected subgradient method in the same way:

$$x^+ = P_C(x - tw(x)) \tag{8.38}$$

for any subgradient w at point x.
The projection P_C is sometimes computationally cheap (or expensive), depending on the set C. For instance, projecting onto a Polyhedron $\{x : Ax \leq b\}$ has no closed form solution, so we need to solve a quadratic problem 8.35 at each step, which is expensive.

8.3.2 Proximal minimization

Consider the following problem for h convex (but not necessarily differentiable):

$$\min h(x) \tag{8.39}$$

The proximal gradient update step is just:

$$x^+ = \text{prox}_t(x) = \underset{z}{\text{argmin}} \frac{1}{2t}\|x - z\|_2^2 + h(z) \tag{8.40}$$

which is faster than subgradient method, but not implementable unless we know $\text{prox}_t(x)$ in closed form.

8.3.3 What do we do if we can't evaluate proximal operator?

In general, all bets are off (in terms of accuracy/ convergence) if you can only do the minimization in the proximal operation *approximately*. If you can precisely control the errors in the approximation, then you can recover the original convergence rates. In practice, this should be done to fairly high precision. See 2011 Schmidt et. al. for related work.

8.4 Accelerated Proximal Gradient Method

Accelerated proximal gradient descent looks like regular proximal gradient descent, but we have changed the argument passed to the *prox* operator and the step at which the gradient update is made with respect to g.

As before, the problem is: min $g(x) + h(x)$, with g convex.

In regular gradient descent, pass $x^{(k-1)} - t_k \nabla g(x^{(k-1)})$ to the *prox* operator and move on.
In accelerated proximal gradient descent, take the last iterate $x^{(k-1)}$ and add a momentum term. Instead of just evaluating the prox at $x^{(k-1)}$, this allows for some of the history to push us a little bit further.

Choose initial point $x^{(0)} = x^{(-1)} \in \mathbb{R}^n$ and repeat $(k = 1, 2, 3, \ldots)$

$$v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)}) \tag{8.41}$$

$$x^{(k)} = \text{prox}_{t_k}(v - t_k \nabla g(v)) \tag{8.42}$$

The choice of constant $\frac{k-2}{k+1}$ is very important for the converge of the algorithm. Thus, this is taken as a fixed constant (which has to/should be tuned). Although there are other convergence strategies, this is provably convergent.

At the first step, when $k = 2$, we are performing regular proximal gradient update.
After that, we are carrying momentum from previous iterations. The momentum weights $\frac{k-2}{k+1} \to 1$ as $k \to \infty$:

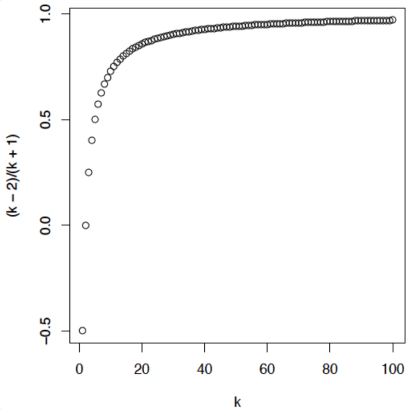

Figure 8.1: Momentum Weights

More momentum is applied later on in the steps of the algorithm. At the start, the current iterate $(x^{(k-1)})$ mostly dictates the next step, and as the algorithm proceeds, we are allowing the history to push us more in the direction we are going.

Why this works (does not tell the whole story):

- At later steps in the proximal descent algorithm, the gradient is going to be small because of proximity to the optimum (progress will be slow close to optimality).
- Acceleration states that if the current iterate is in a good neighborhood of the solution, it's going to continue pushing in that direction, and it it does so more at the later steps of the algorithm (exactly when the gradient is small).

When $h = 0$, this reduces to accelerated gradient descent.

$$v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)}) \tag{8.43}$$

$$x^{(k)} = v - t_k \nabla g(v) \tag{8.44}$$

This is not the same as taking a longer step because the gradient is evaluated at a different spot. Gradient descent converges as a rate of $1/\epsilon$ and the accelerated version converges much faster, thus there must be a bigger difference.

Example: The lasso l_1-penalized problem.

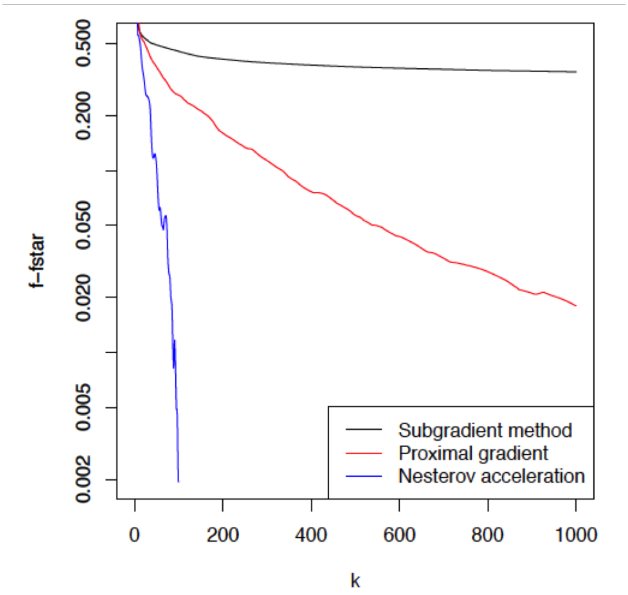

Figure 8.2: Lasso example. Acceleration can really help!

Subgradient method (black): $1/\epsilon^2$ convergence rate, the steps are fairly simple.
ISTA is just proximal gradient descent (red): $1/\epsilon$ converge rate (a lot faster).
FISTA is Nesterov acceleration for gradient descent (blue): $1/\sqrt{\epsilon}$ converge rate.

There are 3 different regimes of convergence that make a big difference in practice.
Observe: the blue curve is not monotonically decreasing and exhibits "Nesterov ripples" $\Rightarrow$ this is not a descent method (not guaranteed that we descend the criterion at every iteration, unlike gradient descent)

8.5 Convergence Analysis

We are minimizing $f(x) = g(x) + h(x)$, assuming the following:

- g: convex, differentiable, $\text{dom}(f) = \mathbb{R}^n$
- ∇g: Lipschitz continuous with constant $L > 0$
- h: convex, prox function can be evaluated

Theorem 8.2 *Accelerated proximal gradient method with fixed step size $t \leq 1/L$ satisfies*

$$f(x^{(k)}) - f^* \leq \frac{2\|x^{(0)} - x^*\|_2^2}{t(k+1)^2} \tag{8.45}$$

Please see Nesterov papers included in the reference list for the proof.
Acceleration achieves the optimal rate $\mathcal{O}(1/k^2)$ for first-order methods (rate of $\mathcal{O}(1/\epsilon^2)$).

8.6 Backtracking Line Search

Backtracking is possible with acceleration, with a strategy similar to what was done in proximal gradient descent. One simple method is as follows:

Fix $\beta < 1$ and $t_0 = 1$
At iteration k start with $t = t_{k-1}$
while $g(x^+) > g(v) + \nabla g(v)^T(x^+ - v) + \frac{1}{2t}\|x^+ - v\|_2^2$
    Shrink $t = \beta t$
else
    Keep x^+
end

Under the same assumptions as before, we obtain the same convergence rate.

Theorem 8.3 *Accelerated proximal gradient method with backtracking line search satisfies*

$$f(x^{(k)}) - f^* \leq \frac{2\|x^{(0)} - x^*\|_2^2}{t_{min}(k+1)^2}, \text{ where } t_{min} = min\{1, \beta/L\} \tag{8.46}$$

One difference between this approach and other strategies is that if we took a certain step size at step k, then the step can only be smaller at step $k+1$. We force ourselves to choose smaller and smaller step sizes across iterations.

8.7 Is Acceleration Always Useful?

It is generally a very effective speed-up tool, but it's not always the thing to turn to. Here are a few things to keep in mind when considering to apply acceleration.

8.7.1 Acceleration Not Necessary

The gain obtained from acceleration is often diminished when solving many optimization problems in sequence and when using "warm starts". In many of these problems here is a penalty parameter, such as in the lasso problem (λ), matrix completion or loss + regularizer problems, among others. For example, when solving lasso problem for tuning parameter values

$$\lambda_1 > \lambda_2 > \ldots > \lambda_r$$

Warm starting:

- When solving for λ_1, initialize $x^{(0)} = 0$ and record the solution $\hat{x}(\lambda_1)$
- When solving for λ_j, initialize $x^{(0)} = \hat{x}(\lambda_{j-1})$, the recorded solution for (λ_{j-1})

The number of iterations required is much smaller when using warm starts as compared to when we do arbitrary initializations. This is because we are already close to the solution when we are starting off. If we believe that the solution is continuous as a function of the regularization parameter λ (true in most cases), then the warm starting is already placing us close to the solution (rather than starting at 0).

In this context, acceleration doesn't do much compared to just running proximal gradient. Thus, warm starting supersedes the need for acceleration (it doesn't hurt, but not worth the trouble).

8.7.2 Acceleration and Backtracking Disadvantageous

Sometimes backtracking and acceleration can be disadvantageous.

Example: Matrix completion problem.
When moving from unaccelerated to accelerated proximal gradient, the prox is evaluated at a different argument.
Proximal gradient update is

$$B^+ = S_\lambda(B + t(P_\Omega - P^\perp(B))) \text{ , where } S_\lambda \text{is the matrix soft-thresholding operator}$$

Here, the proximal operator here is take an SVD and apply soft-thresholding to the singular values.
Problem with backtracking:

- One backtracking loop evaluates the generalized gradient $G_t(x) \Rightarrow$ evaluates $\text{prox}_t(x)$ across multiple values of $t \Rightarrow$ multiple SVDs for matrix completion

Problem with acceleration:

- Acceleration changes the argument passed to the prox operator ($v - t\nabla g(v)$ instead of $x - t\nabla g(x)$). For matrix completion ($t = 1$), this means:

$$B - \nabla g(B) = \underbrace{P_\Omega(Y)}_{\text{sparse}} + \underbrace{P_\Omega^\perp(B)}_{\text{low rank}} \Rightarrow \text{ fast SVD!}$$

$$V - \nabla g(V) = \underbrace{P_\Omega(Y)}_{\text{sparse}} + \underbrace{P_\Omega^\perp(V)}_{\text{not necessarily low rank}} \Rightarrow \text{ slow SVD...}$$

Thus, acceleration is a useful tool, but should not be expected to always yield better performance.

References and Further Reading

Nesterov's four ideas (three acceleration methods):

- Y. Nesterov (1983), "A method for solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$"
- Y. Nesterov (1988), "On an approach to the construction of optimal methods of minimization of smooth convex functions"
- Y. Nesterov (2005), "Smooth minimization of non-smooth functions"
- Y. Nesterov (2007), "Gradient methods for minimizing composite objective function"

Extensions and/or analyses:

- A. Beck and M. Teboulle (2008), "A fast iterative shrinkage-thresholding algorithm for linear inverse problems"
- S. Becker and J. Bobin and E. Candes (2009), "NESTA: a fast and accurate first-order method for sparse recovery"
- P. Tseng (2008), "On accelerated proximal gradient methods for convex-concave optimization"

Helpful lecture notes/books:

- E. Candes, Lecture notes for Math 301, Stanford University, Winter 2010-2011
- Y. Nesterov (2004), "Introductory lectures on convex optimization: a basic course", Chapter 2
- L. Vandenberghe, Lecture notes for EE 236C, UCLA, Spring 2011-2012