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

Lecture 14: October 15 2015

Lecturer: Lecturer: Ryan Tibshirani *Scribes: Benedikt Boecking, Sibi Venkatesan*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

14.1 Dual Cones (previous Lecture)

We recall that a cone $K \subseteq \mathbb{R}^n$ is a set such that $x \in K \Rightarrow tx \in K$ for all $t \geq 0$. Now, the dual cone K^* of K is the set of non-negative dot products of $y \in \mathbb{R}^n$ and $x \in K$. More formally, the dual cone is defined as

$$K^* = \{y \in \mathbb{R}^n : y^T x \geq 0, \forall x \in K\}.$$

Importantly, the dual cone is always a convex cone, even if K is not convex. In addition, if K is a closed and convex cone, then $K^{**} = K$. Note that $y \in K^* \iff$ the halfspace $\{x \in \mathbb{R}^n\}$ contains the cone K. Figure 14.1 provides an example of this in $\mathbb{R}^2$.

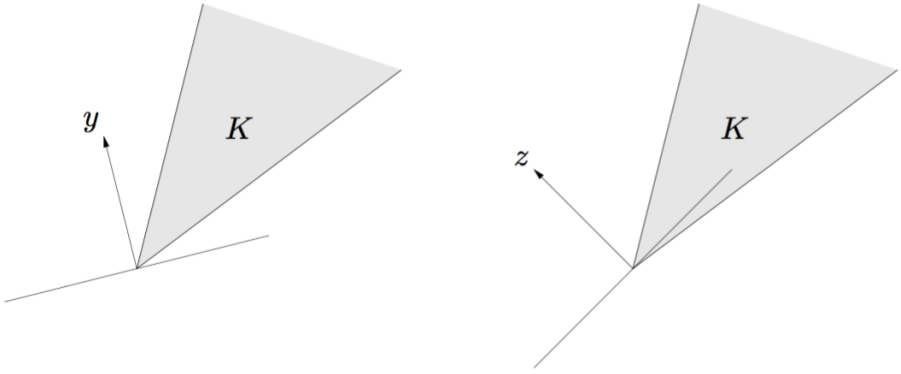

Figure 14.1: When $y \in K^*$ the halfspace with inward normal y contains the cone K (*left*). Taken from [BL] page 52.

14.1.1 Examples of Dual Cones

- **Linear subspace**: the dual cone of a linear subspace V is its orthogonal complement $V^\perp$. E.g. $(row(A))^* = null(A)$, where $null(A)$ denotes the nullspace of A.
- **Norm cone**: the dual cone of the norm cone

  $$K = \{(x, t) \in \mathbb{R}^{n+1} : ||x|| \leq t\}.$$

  is the norm cone of its dual norm

  $$K^* = \{(y, s) \in \mathbb{R}^{n+1} : ||y||_* \leq s\}.$$

- **Positive semidefinite cone**: the convex cone $\mathbb{S}_+^n$ is a *self-dual*, meaning $(\mathbb{S}_+^n)^* = \mathbb{S}_+^n$ Why? Check that

$$Y \succeq 0 \iff tr(YX) \geq 0 \text{ for all } X \succeq 0$$

14.2 Newton's method

We will start by considering the simple setting of an unconstrained, smooth optimization problem

$$\min_x f(x)$$

where our function f is twice differentiable and the domain of the function is $dom(f) = \mathbb{R}^n$. Recall that gradient descent chooses an initial point $x^{(0)} \in \mathbb{R}^n$ and repeats the following

$$x^{(k)} = x^{(k-1)} - t_k \nabla f(x^{(k-1)}), k = 1, 2, 3, ...$$

i.e. it moves in the direction of the negative gradient. In comparison, Newton's method repeats similar steps with the crucial difference that it moves in the direction of the negative inverse of the Hessian times the gradient

$$x^{(k)} = x^{(k-1)} - (\nabla^2 f(x^{(k-1)}))^{-1} \nabla f(x^{(k-1)}), k = 1, 2, 3, ...$$

Note, that there is no notion of a step size in the above definition. This is often referred to as pure Newton's method. Further, note that we can avoid calculating the inverse in each iteration by solving a linear system to obtain the direction $v^{(k)}$

$$\nabla^2 f(x^{(k-1)}) v^k = \nabla f(x^{(k-1)}).$$

We can then define the updates in terms of $v^{(k)}$, namely

$$x^{(k)} = x^{(k-1)} - v^k.$$

Newton's method can be interpreted as doing a better quadratic approximation than gradient descent where the quadratic term uses the actual Hessian

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x).$$

Minimizing this second order Taylor expansion at y yields the Newton update. Figure 14.2 Shows a comparison of gradient descent and Newton's method.

14.2.1 Linearized optimality condition

Another way of interpreting Newton's method is to solve the linearized first order optimality condition. We seek the direction v at step x such that $\nabla f(x + v) = 0$ for a smooth and convex f. Now consider linearizing this condition via a first order Taylor expansion on the gradient equation (think of x being fixed)

$$0 = \nabla f(x + v) \approx \nabla f(x) + \nabla^2 f(x)(x + v - x) = \nabla f(x) + \nabla^2 f(x) v$$

and solving for v, which yields $v = -(\nabla^2 f(x)^{-1} \nabla f(x)$.

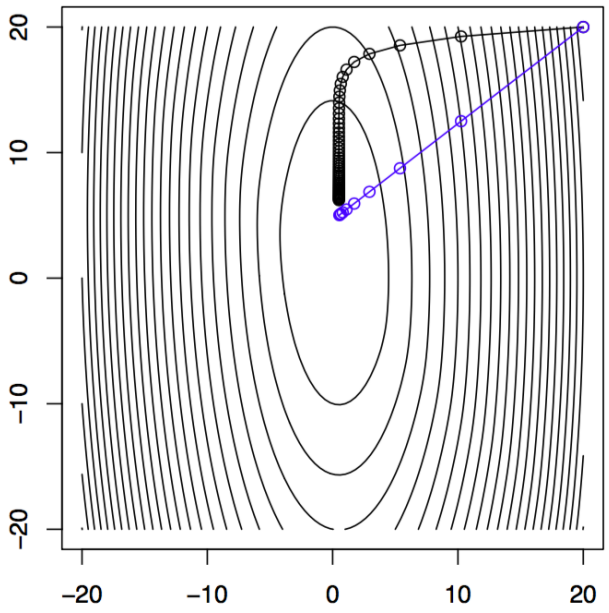

Figure 14.2: This figure compares update steps of gradient descent (black) to Newtons method (blue) for the function $f(x) = (10x_1^2 + x_2^2)/2 + 5log(1 + e^{-x_1 - x_2})$. Each point in this figure is a successive update. Notice that gradient descent takes many more steps than Newton's method, that the update directions are different, that there is a big difference in cost for each update, and that gradient descent moves orthogonal to the contour lines.

14.2.2 Affine invariance of Newton's method

An important property of Newton's method is the *affine invariance* of the Newton step. In the following example we will consider linear scaling. Given f, a non-singular matrix $A \in \mathbb{R}^{n \times n}$, let $x = Ay$ and $g(y) = f(Ay)$. The Newton steps on g are

$$\begin{aligned} y^+ &= y - \left(\nabla^2 g(y)\right)^{-1} \nabla g(y) \\ &= y - \left(A^T \nabla^2 f(Ay) A\right)^{-1} A^T \nabla f(Ay) \\ &= y - A^{-1} \left(\nabla^2 f(Ay)\right)^{-1} \nabla f(Ay) \end{aligned}$$

Note that we achieve the above simply by applying the chain rule twice. Thus we get

$$Ay^+ = Ay - (\nabla^2 f(Ay))^{-1} \nabla f(Ay)$$

i.e.

$$x^+ = x - (\nabla^2 f(x))^{-1} \nabla f(x)$$

Above we have shown invariance under a linear transformation, but if the transformation were affine, i.e. $x = Ay + b$, we would come to the same invariance conclusion. Note that *affine invariance* of the Newton step means that progress is independent of problem scaling. We recall that this is not true for gradient descent.

14.2.3 Newton decrement

The *Newton decrement* at a point x is

$$\lambda(x) = (\nabla f(x)^T (\nabla^2 f(x))^{-1} \nabla f(x))^{1/2}$$

and it relates to the difference between $f(x)$ and the minimum of its quadratic approximation:

$$\begin{aligned} f(x) - \min_y \left(f(x) + \nabla f(x)^T (y-x) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) \right) \\ = f(x) - \left(f(x) - \frac{1}{2} \nabla f(x)^T (\nabla^2 f(x))^{-1} \nabla f(x) \right) \\ = \frac{1}{2} \lambda(x)^2 \end{aligned}$$

We can therefore think of $\lambda(x)^2/2$ as an approximate bound on the suboptimality gap $f(x) - f^*$.

Another interpretation of the Newton decrement uses the update direction of the newton step. If the Newton direction is $v = -(\nabla^2 f(x))^{-1} \nabla f(x)$, then

$$\lambda(x) = \left(v^T \nabla^2 f(x) v \right)^{1/2} = ||v||_{\nabla^2 f(x)}$$

meaning that λ is the *length of the Newton step* in the norm defined by the Hessian $\nabla^2 f(x)$. Note that the Newton decrement, like the Newton steps, are affine invariant; i.e., if we were to define $g(y) = f(Ay)$ for a nonsingular matrix A, then $\lambda_g(y)$ would match $\lambda_f(x)$ at $x = Ay$.

14.3 Backtracking line search

Pure Newton's method doesn't necessarily converge. As described below, Newton's method has a very fast convergence rate. This means that it is also possible for Newton's method to diverge rapidly. If we just take full Newton steps, $x^+ = x - (\nabla^2 f(x))^{-1} \nabla f(x)$, it is possible to diverge depending on where you start.

Due to this, pure Newton's method is very rarely used in practice. Typically, Newton's method also uses backtracking line search in a very similar fashion to gradient descent. We pick two parameters, $\alpha \in (0, \frac{1}{2}]$ and $\beta \in (0, 1)$ and perform very similar updates.

At each Newton iteration, initialize $t = 1$, $v = -(\nabla^2 f(x))^{-1} \nabla f(x)$. While

$$f(x + tv) > f(x) + \alpha t \nabla f(x)^T v$$

shrink $t = \beta t$. Note that $\nabla f(x)^T v = -\lambda^2(x)$. Since this can be pre-computed, all we have to do every iteration is function evaluations along a line.

14.3.1 Example: Newton's method vs Gradient Descent

Here is an example of logistic regression with 500 observations and 100 variables. This is not a quadratic so neither method will converge to an exact solution. The performance of the algorithms (both using backtracking line search) has been shown in the figure below. The plot shows the number of iterations vs. the difference between current objective value and minimum. The actual minimum was using R's GLM

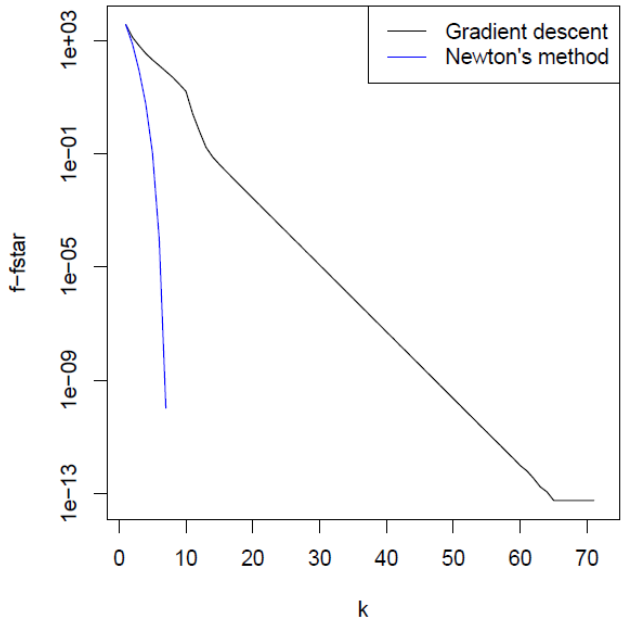

Figure 14.3: Newton's Method vs. Gradient Descent. Taken from Ryan Tibshirani's slides.

package which uses Newton's method itself. Newton's method reaches converges after around 7 iterations, while Gradient Descent reaches around the same accuracy after 65-70 iterations. Newton's method has "quadratic" convergence, as described below.

Note: It is a bit unfair to compare these methods based on number of iterations, as the cost of each iteration is very different. For Newton's method, we need to compute the Hessian, gradient and solve a linear system of equations. For Gradient Descent, we only need the gradient.

14.4 Convergence Analysis

Assume that we have a convex and twice-differentiable function f, where the domain of f is $\mathbb{R}^n$. Further, we assume that:

- ∇f is Lipschitz with parameter L
- f is strongly convex with parameter m
- $\nabla^2 f$ is Lipschitz with parameter M
  Given that $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, this means that for all x, y, we have $||\nabla^2 f(x) - \nabla^2 f(y)||_F \leq M||x - y||_2$.

The first two conditions above guarantee that Gradient Descent will have a linear convergence rate: it will converge as c^k where k is the number of iterations, for $c < 1$. But c depends adversely on the condition number of the hessian: $\frac{L}{m}$. If this is large, then c is close to 1, and convergence of Gradient Descent will be very slow.

The following is the convergence result for Newton's method using backtracking line search for a function f which satisfies the above properties.

$$f(x^{(k)}) - f^\star \leq \begin{cases} (f(x^{(0)}) - f^\star) - \gamma k & \text{if } k \leq k_0 \\ \\ \dfrac{2m^3}{M^2}\left(\dfrac{1}{2}\right)^{2^{k-k_0+1}} & \text{if } k > k_0 \end{cases}$$

where α, β are the backtracking line search parameters, $\gamma = \dfrac{\alpha\beta^2\eta^2 m}{L^2}$, $\eta = \min\{1, 3(1-2\alpha)\}\dfrac{m^2}{M}$, and k_0 is the number of steps until $||\nabla f(x^{(k_0+1)})||_2 < \eta$.

Thus, Newton's method converges in two stages, determined by some number of iterations k_0. The first stage of convergence is called the damped phase. It tells us that every iteration, we move γ closer to the optimum. The second stage of convergence is called the pure phase. Convergence is extremely quick, and is known as quadratic convergence.

Note that the bounds for convergence get worse the more poorly conditioned the function. If the condition number if very large, then γ would be tiny. This means that we would not expect to make much progress in each iteration during the first stage.

Here are a few more details on the different phases, given $\gamma > 0$ and $0 < \eta \leq \dfrac{m}{M^2}$.

- **Damped phase:** $||\nabla f(x^{(k)})||_2 \geq \eta$ and

  $$f(x^{(k+1)}) - f(x^{(k)}) \leq -\gamma$$

  Backtracking line search in this phase will typically select a step-size of less than 1. Thus, this phase is referred to as damped.

- **Pure phase:** $||\nabla f(x^{(k)})||_2 < \eta$ and

  $$\frac{M}{2m^2}||\nabla f(x^{(k+1)})||_2 \leq \left(\frac{M}{2m^2}||\nabla f(x^{(k)})||_2\right)^2$$

  Backtracking line search will exit immediately when $t = 1$, and thus it is called pure. Once we enter the pure phase, we won't leave, since when $\eta \leq \dfrac{m^2}{M}$:

  $$\frac{2m^2}{M}\left(\frac{M}{2m^2}\eta\right)^2 < \eta$$

Let's unravel this result a bit. Say we want to reach $f(x^{(k)}) - f^\star \leq \epsilon$. The overall number of iterations needed to converge to this accuracy is:

$$\frac{f(x^{(0)}) - f^\star}{\gamma} + \log\log\left(\frac{\epsilon_0}{\epsilon}\right)$$

To enter the pure Newton phase, we need at most $\dfrac{f(x^{(0)}) - f^\star}{\gamma}$ iterations. This is easy to see as if we were to take more than this, then the difference $f(x^{(k)}) - f^\star$ would be negative which is not possible.

Once we are in the pure phase, we can see that the convergence rate is $\log\log\left(\dfrac{\epsilon_0}{\epsilon}\right)$, where $\epsilon_0 = \dfrac{2m^3}{M^2}$. This is vastly different from $\log\dfrac{1}{\epsilon}$ for Gradient Descent. However, this is only a local convergence rate. We need a certain number of iterations to enter this phase to begin with. If we were to look at a global convergence rate, then Newton's rate might still have only a linear convergence rate.

14.4.1 Self-concordance

The above bound for convergence still depends on the Lipschitz and strong-convexity parameters of the problem: L, m, M. But the algorithm itself does not depend on these, as it is affine invariant. If we had a poorly conditioned Hessian, we could potentially transform it into a coordinate system where it was no longer poorly conditioned. Thus, the bound was bothersome since it represented a gap between theory and practice. The concept of self-concordance allows us to bridge this gap and was thus a very big deal when it was discovered.

A convex function f on $\mathbb{R}$ is called self-concordant if:

$$|f'''(x)| \leq 2f''(x)^{3/2} \text{ for all } x$$

For example consider $f(x) = -\log(x)$. Then $f'(x) = -\frac{1}{x}$, $f''(x) = \frac{1}{x^2}$ and $f'''(x) = -\frac{2}{x^3}$. It's easy to check that this is indeed self-concordant.

The analysis of Newton's method for self-concordant functions is scale-free; it does not depend on the scaling parameters of the problem. Here is the relevant result from Nesterov and Nemirovskii. Newton's method on self-concordant functions with backtracking requires at most

$$C(\alpha, \beta)(f(x^{(0)}) - f^\star) + \log\log\frac{1}{\epsilon}$$

iterations to reach $f(x^{(0)}) - f^\star \leq \epsilon$. Here, $C(\alpha, \beta)$ is a constant that only depends on the backtracking parameters. This result is taken from a book on second-order methods and interior point methods by Nesterov and Nemirovskii [NN].

14.5 Comparison to First-Order methods

Here are some high-level comparisons.

- **Memory:** Each iteration of Newton's method requires $O(n^2)$ storage, as we have to store the $n \times n$ Hessian. We cannot store this in memory as it changes every iteration. Gradient Descent only requires $O(n)$ storage of the gradient at every iteration.
- **Computation:** In each iteration of Newton's method, we need to solve a linear system which takes $O(n^3)$ flops in general. Each Gradient Descent iteration only requires $O(n)$ flops for simple vector operations.
- **Backtracking:** This has roughly the same cost across the two methods: both require $O(n)$ flops. We can compute the update step for Newton's method before we begin backtracking.
- **Conditioning:** In principle Newton's method is not affected by a problem's conditioning: Any affine transformation will leave the steps unchanged. Gradient Descent can perform poorly if the problem does not have good conditioning.
- **Fragility:** This is basically the issue of numerical stability. Newton's method is more susceptible to numerical errors since solvers are usually sensitive to conditioning. Thus, Newton steps could be noisy or erroneous. Gradient Descent is generally more robust to stability issues.

14.5.1 Example Revisited: Newton's method vs Gradient Descent

Here is the same example of the two methods on logistic regression. The plot now shows time on the x-axis instead of the number of iterations, while y-axis remains the same.

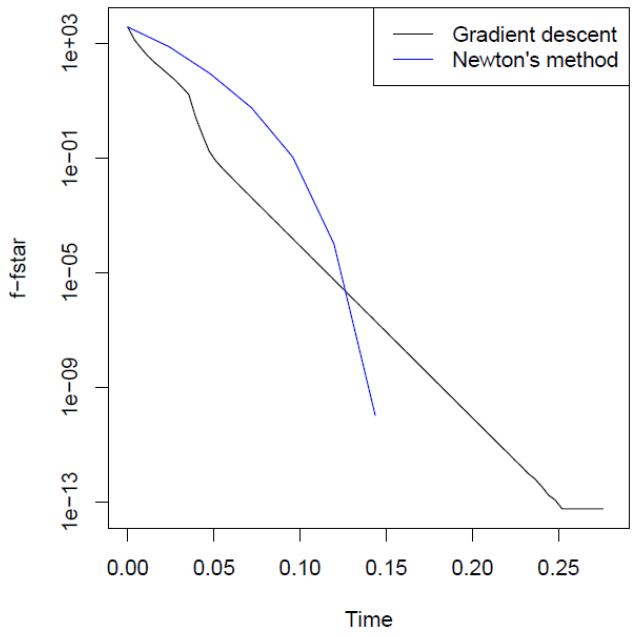

Figure 14.4: Newton's Method vs. Gradient Descent. Taken from Ryan Tibshirani's slides.

Each Newton step takes $O(p^3)$ time while each Gradient step takes $O(p)$ time for p variables. So this is slightly more favorable to Gradient Descent. However, we still do see the quadratic convergence behavior for Newton's method, which doesn't exist for Gradient Descent.

14.5.2 Sparse and Structured Problem

There are some situations where Newton's method will perform well. If the Hessian of our function is sparse and structured, and if we can solve linear systems efficiently and reliably, we would want to use Newton's method.

For example, if Hessian is banded, then storage and computation time (solving linear systems) are both $O(n)$ for n variables. This puts Newton's method on the same scale as a first-order method; it does not suffer from some of its main setbacks on general problems.

14.6 Equality-constrained Newton's method

Here is a brief discussion of Newton's method applied to problems with equality constraints. Consider the problem:

$$\min f(x) \text{ subject to } Ax = b$$

There are three things we can do:

- **Eliminate the equality constraints:** We can re-parameterize our problem in terms of the null-space of A, i.e. we write $x = My + x_0$ where M spans the null-space of A and $Ax_0 = b$.

  While this is a reasonable idea, there are a couple of problems. This requires us to find M, a basis for the null-space. Also, it might ruin any structure we previously had in the problem. For example, the Hessian of the re-parameterized problem might not be sparse, even if that of the original problem was.

- **Derive the dual:** If we work with the dual problem, we see that the linear equality constraint gets lifted up into the criterion itself: $-f^*(-A^T v) - b^T v$

  But it's not always that simple as we need to still compute the conjugate function f^*. Further, we need to be able to relate the solutions of the primal and dual problems, x^* and v^*.

- **Equality-constrained Newton:** This is often the most straight-forward option, and is described in a little more detail below.

In equality-constrained Newton's method, we take Newton steps as we did before. But for our step direction, instead of just minimizing the quadratic approximation of f, we add a constraint that we must respect our equality constraint. In particular:

$$x^+ = x + tv$$

$$v = \underset{z:Az=0}{\arg\min} \nabla f(x)^T (z - x) + \frac{1}{2}(z - x)^T \nabla^2 f(x)(z - x)$$

Thus, if we move along v from a feasible point, we will still remain at a feasible point.

From the KKT conditions, we see that this reduces to solving a single linear system as follows:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}$$

Solving this system in v gives us the equality-constrained step direction. This system is often sparse and structured, as the structure in the Hessian is preserved. We could even do some sort of block-decomposition to solve this in a more refined manner.

References

[BL] S. Boyd and L. Vandenberghe, "Convex Optimization," Chapter 4

[NN] Y. Nesterov and A. Nemirovskii, "Interior-point polynomial methods in convex programming," Chapter 2