EE 227A: Convex Optimization and Applications **April 5, 2012**

Lecture 19: **Interior-Point Methods**

Lecturer: Laurent El Ghaoui

Reading assignment: Chapter 9 of BV.

19.1 Setup

Motivation. As we will see later, interior-point techniques for constrained convex minimization rely on approximating the constrained problem

$$\min_x \; f_0(x) \;:\; f_i(x) \leq 0, \;\; i = 1, \ldots, m,$$

where f_i's are smooth, by the smooth, *unconstrained* problem

$$\min_x \; \mu f_0(x) + B(x),$$

where B is a barrier for the feasible set $\mathcal{X}$ of the original problem, that is, a convex function with the property that $B(x) \to +\infty$ as x goes towards the boundary of $\mathcal{X}$ from its interior. One example of a barrier function, which plays a huge role in the so-called interior-point methods, is the logarithmic barrier:

$$B(x) = -\mu \sum_{i=1}^{m} \log(-f_i(x))$$

where $\mu > 0$ is a small parameter.

The parameter μ allows to control how much inside the feasible set we want to be, with respect to how much we insist on minimizing the objective; interior-point algorithms work by solving a sequence of unconstrained problems with increasing values of μ. While there are many choices possible for the barrier function, at this point it is unclear which ones will lead to a computationally efficient (polynomial-time) algorithm.

The above shows that it is important to develop reliable and efficient methods for smooth, unconstrained minimization. The focus of this lecture is one particular method called Newton's method, which is the basic building block of interior-point methods for constrained convex minimization. Newton's method does not always work well, even for convex functions; we will see how a special class of convex functions, *self-concordant* functions, leads to an efficient algorithm. In turn, this will guide our choice for a barrier when we study the constrained case.

Problem statement. In this lecture, we consider the *unconstrained problem:*

$$p^* = \min_x \ f(x), \tag{19.1}$$

with the following assumptions:

- f is convex and twice differentiable;
- p^* is finite and attained.

Minimizing sequence. Since f is differentiable and convex, the optimality condition is given by

$$\nabla f(x) = 0 \tag{19.2}$$

Solving the unconstrained minimization problem (19.1), is the same as finding the solution of (19.2). Generally, this problem must be solved by an iterative algorithm. Our goal is then to devise an algorithm that produces a sequence $x^{(k)}$ such that:

$$f(x^{(k)}) \to p^* \text{ as } k \to \infty.$$

Such a sequence is called a *minimizing sequence* for problem 19.1.

Closed epigraph assumption. The methods that we will discuss require a suitable starting point. In fact, we need to start with the initial point, $x^{(0)}$ such that $x^{(0)} \in \text{dom } f$ and we must assume that the $f(x_0)$-sublevel set is closed, where the α-sublevel set is defined by $\{x \ : \ f(x) \leq \alpha\}$. This condition is in general hard to check, but the stronger condition that *all* sublevel sets are closed is usually easier, since it is equivalent to the fact that the epigraph of f is closed. Examples of twice differentiable functions f with closed epigraphs include

$$f(x) = \log\left(\sum_{i=1}^{n} \exp(a_i^T x + b_i)\right), \quad f(x) = -\sum_{i=1}^{m} \log(b_i - a_i^T x).$$

The last examples plays an important role in interior-point methods for LP.

Strong convexity and implications. The convergence analysis seen later will require the notion of strong convexity.

Definition 1 (Strong Convexity). *A function f is strongly convex on a set S if*

$$\exists m > 0 \ : \ \forall\, x \in \textbf{dom}\, f, \ \ \nabla^2 f(x) \succeq mI. \tag{19.3}$$

Note that strong convexity is stronger than strict convexity.

Strong convexity has several implications. Let $S = \{x : f(x) \leq f(x_0)\}$. When the function f is convex we have

$$\forall x, y \ \in \ S, \ f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{19.4}$$

If the function f is strongly convex we have the stronger condition

$$\forall x, y \ \in S, \ f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2}\|x - y\|_2^2, \tag{19.5}$$

and therefore $p^* > -\infty$. Furthermore, it can also be shown that

$$\forall x \in S, \ 0 \leq f(x) - p^* \leq \frac{1}{2m}\|\nabla f(x)\|_2. \tag{19.6}$$

This gives a stopping criterion, provided we know m (or, a lower bound).

Algorithm format. Our goal is to devise an algorithm of the form:

$$x^+ = x + t\Delta x \tag{19.7}$$

where x^+ is the update, x is the current variable, t is the step size, and Δx is the *step*, or *search direction*. The search direction Δx and the step size t are usually obtained by two separate processes, the latter being called *line search*. We denote by $x^{(k)}$ the iterates produced by the algorithm above.

Once an algorithm is designed, the purpose of convergence analysis is to find a bound on the number of iterations k needed to achieve ϵ-convergence, that is k such that $p^* \leq f(x^{(l)}) \leq p^* + \epsilon$ for every $l \geq k$.

19.2 Descent Methods

19.2.1 General descent methods

If there exists t such that

$$f(x^+) < f(x),$$

then Δx is called a *descent direction*. A *descent method* is one that uses descent directions at each step.

Note that from convexity, $\nabla f(x)^T(y-x) \geq 0$ implies $f(y) \geq f(x)$, so the search direction in a descent method must satisfy $\nabla f(x)^T(\Delta x) < 0$. In other words, it must make an acute angle with the negative gradient.

Descent method: algorithm. The general descent method is as follows. Given a starting point, $x \in \textbf{dom}\, f$, repeat the following steps:

1. Determine a descent direction Δx.
2. Line Search: Choose a step size t.
3. Update: $x^+ = x + \Delta x$.

and stop when the stopping criterion is satisfied.

Line search. The second step (the line search) can be performed by exact or inexact methods. In *exact* line search, the step size t is chosen to minimize function f along the ray $\{x + t\Delta x : t \geq 0\}$, that is:

$$t^* = \arg\min_{t>0} f(x + t\Delta x).$$

Note that this is a convex problem in one variable.

In *inexact* line search, a technique called *backtracking* is used. This method depends on two constants, $\alpha \in (0, 1/2)$ and $\beta \in (0, 1)$. We set $t = 1$ initially and then we repeat $t := \beta t$ until we obtain

$$f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^T \nabla x.$$

The backtracking exit inequality holds for some $t \geq 0$ in an interval $(0, t_0]$. The main cost of this algorithm is just function evaluations. This method is easier to implement than the exact line search method.

19.2.2 Gradient descent method

As a specific example of a descent method, consider the *gradient method,* which uses the negative gradient as the search direction:

$$\Delta x = -\nabla f(x).$$

This method has the following characteristics:

- (Usual) stopping criterion: $\|\nabla f(x)\|_2 \leq \varepsilon$.
- Convergence: for strongly convex f we have:

  $$0 \leq f(x^{(k)}) - p^* \leq c^k (f(x^{(0)}) - p^*),$$

  where $c \in\ (0, 1)$ depends on $m,\ x^{(0)}$, and the line search parameters. This shows that $f(x^{(k)})$ converges to p^* as $k \to \infty$

In practice however, this method can be very slow.

19.3 Newton Methods

Newton's method is a particular method that relies on the Hessian of the function at each step. It is thus part of a class of methods known as second-order methods. This is in contrast with say the gradient method, which is a first-order method.

19.3.1 Step and decrement

Newton step. For $x \in \text{dom } f$, the *Newton step* is the step defined as:

$$\Delta x_{NT} = -(\nabla^2 f(x))^{-1} \nabla f(x) \tag{19.8}$$

(Remember, $(\nabla^2 f(x))^{-1}$ exists because f is assumed to be strongly convex.)

The second order Taylor approximation of $f(x+v)$ is:

$$f(x+v) \approx \hat{f}(x+v) := f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla_2 f(x) v \tag{19.9}$$

which is a convex quadratic function of v, which is minimized when $v = \Delta x_{NT}$. Thus, the update $x + \Delta x_{NT}$ minimizes the second order approximation of f.

Newton decrement. The quantity

$$\lambda(x) = \sqrt{\nabla f(x)^T (\nabla^2 f(x))^{-1} \nabla f(x)} \tag{19.10}$$

is called the *Newton decrement* at x. This measures the proximity to x^*. (In other words, it provides the quality of the method's estimates.)

We can relate $\lambda(x)$ to the quantity $f(x) - \inf_y \hat{f}(y)$, where $\hat{f}$ is the second-order approximation of f at x, and $\hat{f}(y)$ is equal to $\hat{f}(x+v)$ evaluated at $v = y - x$:

$$0 \le f(x) - p^* \approx f(x) - \inf_y \hat{f}(y) = \frac{1}{2}\lambda(x)^2 \tag{19.11}$$

Thus, $\lambda^2/2$ is an estimate of $f(x) - p^*$ based on the quadratic approximation of f at x. Furthermore note that $\lambda(x)$ is affinely invariant, meaning that if we change coordinates in an affine fashion, the Newton decrement stays the same.

19.3.2 Algorithm description

Newton's method is a descent method with

$$\Delta x = \Delta x_{NT}.$$

The algorithm is as follows: given a starting point $x \in$ *domf*, and tolerance ε,

1. Find the step: Compute $(\Delta x)_{NT}$ and $\lambda(x)$.
2. Stopping criterion: stop if $\lambda^2/2 \le \varepsilon$.
3. Line search: Choose step size t.
4. Update: $x = x + t(\Delta x)_{NT}$.

Newton's method satisfies a useful property: it is *affinely invariant.* By invariance we mean $\forall T \in \mathbf{R}^n$, T invertible, t Newton's methods iterates (y^k) for the function $\tilde{f}$ with values $\tilde{f}(y) = f(Ty)$ are related to the iterates (x^k) for f via the affine transformations

$$y^k = T^{-1} x^k, \;\; k = 0, 1, 2, \ldots$$

19.3.3 Classical convergence analysis

The classical convergence analysis relies on the following assumptions:

- f is strongly convex, with constant $m > 0$.
- $\nabla^2 f$ is Lipschitz continuous with constant $L > 0$: that is, for every $x, y \in \mathbf{dom}\, f$, we have $\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L\|x - y\|_2$.

If we fulfill these two assumptions (which are generally hard to check), then there exist constants $\eta \in (0, \frac{m^2}{L}]$ and $\gamma > 0$ such that:

- if $\|\nabla f(x)\|_2 \geq \eta$, then $f(x^{k+1}) - f(x^k) \leq -\gamma$ (Phase 1);
- if $\|\nabla f(x)\|_2 \leq \eta$, then $\frac{L}{2m^2}\|\nabla f(x^l)\|_2 \leq (\frac{1}{2})^{2^{l-k}}$ (Phase 2).

Phase 1 requires at most $\frac{f(x^0)-p^*}{\gamma}$ iterations, whereas Phase 2 requires at most $\log\log(\varepsilon_0/\varepsilon)$ iterations. So the total number of iterations to attain $f(x) - p^* \leq \varepsilon$ is bounded above by:

$$\frac{f(x^0) - p^*}{\gamma} + \log\log(\varepsilon_0/\varepsilon)$$

where ε_0 and γ depend on x^0, m, and L.

19.3.4 Newton's method for self-concordant functions

Flaws of classical convergence analysis. There are two major problems with the classical convergence analysis of Newton's method:

1. It depends on unknown constants $(m, L, \dots)$, which are rarely known in practice.
2. The bound of the number of steps required to reach convergence is not affinely invariant, while Newton's method is.

These problems make it hard to obtain reliable estimates of the computational effort required to solve a given problem.

Self concordance. In contrast, convergence analysis for self-concordant functions has the following advantages:

- It does not depend on unknown constants.
- It gives an affinely-invariant bound.

Definition 2 (Self-Concordance). *A function $f : \mathbf{R} \to \mathbf{R}$ is self-concordant if:*

1. *f is convex, three times differentiable.*
2. *For every $x \in \mathbf{dom}\, f$, $|f'''(x)| \le 2(f''(x))^{3/2}$.*

A function $f : \mathbf{R}^n \to \mathbf{R}$ is self-concordant if for every v, the function $t \to f(x + tv)$ is self concordant.

Self-concordance is a condition that imposes that the Hessian of the function does not vary too fast in the metric induced by the Hessian itself. It can be compared to the Lipschitz continuity condition on the Hessian seen earlier.

A self-concordant function is preserved under scaling (with scaling factor $\alpha \ge 1$) and addition. Furthermore, the property of self-concordance is affinely invariant.

Self-concordance is usually hard to check, however a surprisingly large number of important barrier functions satisfy this property.

Examples. The following self-concordant functions are the essential building blocks for interior-point methods in conic programming:

- $f(x) = -\sum_{i=1}^m \log x_i$ (for LP).
- $f(x) = -\log(t^2 - x^T x)$, where $t > \|x\|_2$ (for SOCP).
- $f(x) = -\log\det X$, where $X = X^T \succ 0$ (for SDP).

Specifically, consider unconstrained problems of the form

$$f(x) = \mu c^T x - \sum_i \log(b_i - a_i^T x),$$

where $\mu > 0$, and which will be used to solve LPs of the form

$$\min c^T x : a_i^T x \le b_i, \quad i = 1, \dots, m.$$

If μ is very small, we mostly care about being in the interior of the feasible set. If μ is very large, our main concern is to minimize the objective. As $\mu \to +\infty$, the optimal solution to the unconstrained problem above tends towards a solution to the original LP.

Due to the affine invariance of the self-concordance property, the function above is self-concordant. Therefore the number of iterations to solve each unconstrained problem does not depend on the size of the problem. However, each iteration's complexity does; furthermore, there is a small extra price to pay as we have to solve the problem for several values of the parameter μ. We return to this later.

Number of iterations needed for convergence. For self-concordant functions the number of iteration needed to achieve ϵ-convergence is:

$$\frac{f(x^{(0)}) - p^*}{\gamma} + \log\log(1/\varepsilon),$$

where γ depends on the backtracking parameter. With typical values of these parameters, and $\epsilon = 10^{-12}$, we obtain the bound $375(f(x^0) - p^*) + 6$.

Note that this is only a (crude) upper bound on the number of iterations actually needed in practice. The practical number is more like a constant (between 10 and 50 iterations), independent of problem size, starting point, etc.

Note that in the above formula, we do not know $p*$, but later we will use duality to bound p^* from below. In contrast with the classical convergence analysis of Newton's method, the above bound is independent of Lipschitz constants or strong convexity parameters.

To summarize, Newton method for self-concordant functions needs a number of iterations independent of problem size, and extremely moderate dependence on precision ϵ. Each iteration of the algorithm requires solving a system of the form $Hx = g$, where g is the negative gradient, and $H \succeq 0$ is the Hessian. In practice, this computation of the Newton step is the main source of computational effort. For dense Hessians, the complexity of each Newton step is $O(n^3)$. Since the number of such iterations is constant, the method has an overall complexity of $O(n^3)$ as well.

19.3.5 Practical complexity

Solving the Newton system. Let us introduce the Cholesky factorization of H: $H = LL^T$, with L lower triangular; that decomposition costs about n^3 flops. Then, $\Delta x_{NT} = L^{-1^T} L^{-1} g$ can be obtained by solving two successive triangular systems, while the Newton decrement is $\lambda(x) = \|L^{-1}g\|_2$.

Exploiting structure: examples. For specific problems, it is often possible to exploit problem structure and lower the complexity of the method.

For example, if the function f has the form

$$f(x) = \sum_{i=1}^{n-1} \psi_i(x_{i+1}, x_i),$$

where ψ_i's are two-variable functions, the resulting Hessian will be tridiagonal, and the complexity of each Newton step is linear in n.

As another example, consider the case when

$$f(x) = \sum_{i=1}^{m} \psi_i(x_i) + \psi_0(Ax + b),$$

where $\psi_0, \ldots, \psi_m$ are strongly convex, twice differentiable, and $A \in \mathbf{R}^{p \times n}$, with $p << n$. The above problem is almost decoupled. We have

$$H = D + A^T H_0 A,$$

where D is diagonal and H_0 is the $p \times p$ Hessian of ψ_0.

We can solve the system efficiently, as follows. First form the Cholesky factorization of $H_0 = L_0 L_0^T$. We write the Newton system as

$$D\Delta x + A^T L_0 w = -g, \quad L_0^T A \Delta x = w,$$

where w is a new dummy variable. Now eliminate Δx from the first equation, then compute $w, \Delta x$ from

$$(I + L_0^T A D^{-1} A^T L_0) w = -L_0^T A D^{-1} g, \quad D\Delta x = -g - A^T L_0 w.$$

The cost is now $O(np^2)$, which is much smaller than the cost of $O(n^3)$ we would get without exploiting the structure of the problem.