STAT 538 Lecture 3
January 17, 2012
Unconstrained optimization – Part I
©Marina Meilă
mmp@stat.washington.edu

Reading: For mathematical background, Appendix A of Boyd & Vandenberghe (BV). For Unconstrained optimization proper, BV Ch 9. The notes follow mostly Bertsekas (B) Chapter 1. For the line minimization methods, look at B (ch 1), at "Numerical recipes" (NR) chapter 10. NW (Ch 3, 6) is also useful reading for both line minimization and Quasi-Newton.

1 Overview

Problem Find $\min_x f(x)$ for $x \in \mathbb{R}^n$ or $x \in D$ the **domain** of f. We will assume also that f is a twice differentiable function with continuous second derivatives.

Notation The **gradient** of f is the column vector

$$\nabla f(x) \;=\; \left[\frac{\partial f}{\partial x_i}(x)\right]_{i=1}^{n} \tag{1}$$

and the **Hessian** of f is the square symmetric matrix of second partial derivatives of f

$$\nabla^2 f(x) \;=\; \left[\frac{\partial^2 f}{\partial x_i x_j}(x)\right]_{i,j=1}^{n} \tag{2}$$

A **local minimum** for f is point x^* for which

$$f(x^*) \;\leq\; f(x) \quad \text{whenever } ||x - x^*|| \;<\; \epsilon$$

A **global minimum** for f is point x^* for which

$$f(x^*) \;\leq\; f(x) \quad \text{for all } x \text{ in the domain of } f$$

We say x^* is a **strict local/global minimum** when the above inequalities are strict for $x \neq x^*$. A minimum is **isolated** if it is the only local minimum in an ϵ-ball around itself.

A **stationary point** for f is a point x^* for which $\nabla f(x^*) = 0$.

Proposition If f has continuous second derivative everywhere in D, and $x^* \in D$ is a point for which $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \geq 0$ ($\nabla^2 f(x^*) > 0$) then x^* is a **(nonsingular)** local minimum for f.

In what follows, we will deal only with non-singular local minima. A non-singular local minimum is *strict* and *isolated*.

In the analysis of algorithms and practically, it is important to know if f has a **finite global minimum**; this is equivalent with f being bounded below by a constant. Otherwise, the global minimum of f is $-\infty$ and the optimization algorithms will not converge on this problem (or will converge to other local minima).

Many unconstrained optimization methods for finding a local minimum are of the form:

$$x^{k+1} \;=\; x^k + \alpha^k d^k \tag{3}$$

where $d^k \in \mathbb{R}^n$ represents an **(unnormalized) direction** and $\alpha^k > 0$ is a scalar called the **step size**.

Direction choice

- gradient based $d^k = -D^k \nabla f(x^k)$ with $D^k \in \mathbb{R}^{n \times n}$
  - steepest descent $D^k = I$
  - stochastic gradient (more about it later)
  - Newton-Raphson $D^k = \nabla^2 f(x^k)^{-1}$
  - conjugate gradient – implicity multistep rescaling of the axes "equivalent" to $D^k = \nabla^2 f(x^k)^{-1}$
  - quasi-Newton – implicit multistep approximation of $D^k = \nabla^2 f(x^k)^{-1}$
- non-gradient based
  - coordinate descent $d^k =$ one of the basis vectors in $\mathbb{R}^n$

Step size choice

- line minimization $\alpha^k = \min_\alpha f(x^k + \alpha d^k)$
- Armijo rule (also called Backtracking) = search but not minimization
- constant step size $\alpha^k = s$
- diminishing step size $\alpha^k \to 0$; $\sum_k \alpha^k = \infty$

2 How to evaluate an optimization method?

- Does it converge to a minimum?
- How fast?
- Practical issues: Is it easy to implement or tune? Available software?

As we shall see, all the methods described here converge to a minimum, but some of the require the function f to have additional "good" properties.

For the second question, the answer is usually given in terms of **rates of convergence**, because of the general assumption that we'll use an iterative algorithm to find the minimum.

Let $e^k = x^k - x^*$ or $e^k = f(x^k) - f(x^*)$ denote the "error" at step k. Then, an algorithm has a rate of convergence of order p if

$$||e^{k+1}|| \leq \beta(||e^k||)^p \quad \text{for some } 0 < \beta < 1 \tag{4}$$

In the above, $p > 0$ but not necessarily an integer. However, the most common cases are $p = 1$ (**linear**) and $p = 2$ (quadratic). A rate of $p < 1$ is possible but is considered too slow in practice. Superlinear scales are desirable – and often achievable.

Note that the use of the term "linear" here is inconsistent with its use in e.g complexity theory. If an optimization algorithm is *linear*, that means that the error decreases *exponentially* with k, as $||e^k|| \leq \beta^k ||e^0||$.

In optimization problems, there are various ways of expressing the computational complexity of an algorithm:

- *number of flops* (floating point operations) per iteration, usually as a function of n the dimension of the problem

- *number of function* or gradient evaluations per iteration
- *number of iterations*; this latter quantity is given implicitly, by the rate of convergence.
- *memory requirements*
- With the increased complexity and variation of computer systems, the above mentioned number of operations is becoming obsolete. Algorithms are increasingly judged by other, system-related qualities, like: type of memory access (do they access memory in blocks or randomly), cache misses, etc. These criteria are beyond the scope of this course, but what you need to remember is that the texbook properties on an algorithm alone do not always predict its performance on the system you are going to run it. You may need to experiment with parameters and with algorithms to determine which algorithm is better suited for your data and system.

There are two regimes for each algorithm:

- the transitory, or approach regime, when x^k is far away from x^*
- the asymptotic regime, near x^* – most classic results are about this regime

The theoretical (and practical) behavior of optimization algorithms will strongly depend on the properties of the function f to be optimized around its minimum x^*. Typically, we will assume that the function is twice differentiable and that its Hessian $\nabla^2 f$ is continuous and strictly positive definite around x^*. But other, weaker, conditions on f also indicate an f that is "easy" to minimize. Here are two of the most common ones.

We say that the problem $\min_x f$ is a **smooth** minimization problem if f it is *upper bounded* by a quadratic function around x^*, i.e. iff there exists $M > 0$ so that

$$f(x) - f(x^*) \leq \frac{1}{2} M ||x - x^*||^2$$

on a neighborhood of x^*. This property indicates that f, even though it may not be differentiable, behaves "almost like a quadratic", in the sense that local quantities (gradient, Hessian) are informative w.r.t the minimum. An example of a non-smooth minimization problem is $\min_x |x|$. The gradient

∇f is ± 1 everywhere but in 0, so it gives us information on which side of x the minimum lies, but its size does not tell us how far we are from x^*.

Note that the term smooth above refers to the *minimization problem* and *not to the function* f, which may not be smooth itself. [Exercise: Prove that if f has a continuous Hessian around x^* and if $\nabla^2 f(x^*) \succ 0$ the problem is smooth.]

Another criterion that distinguishes easier problems is that of **strongly convexity** of f. Essentially, f is strongly convex when it is *lower bounded* by a quadratic function (more about this when we study convexity). This property indicates that the minimum x^* is "well isolated", or, in other words, if $f(x)$ is close to the optimum, then x will also be close to x^*. Hence, this property lets us calculate the precision in x from the (easier to obtain) precision in f.

3 Line minimization algorithms

3.1 Line minimization by the Golden Section Rule

(See also NR or NW)

1. Bracket the minimum. Find an interval $[0, s]$ that contains the desired α. This can be done by iteratively doubling s until $f(x^k + sd^k) > f(x^k)$.

   A more general definition of a bracketed minimum (not assuming that the function is decreasing at x^k in the direction of d^k is to find a triplet of points on the line $a = x^k$, b, c with b between a and c and $f(b) < f(a), f(c)$. NR gives a method of finding such a triplet by starting with an initial value for b and iteratively expanding the candidate triplet.

2. Find the minimum in [0, s]; this is the Golden Section method proper

R the golden number is the positive root of the equation

$$\frac{R}{1} = \frac{1-R}{R} \tag{5}$$

The reason is given by the figure below: we want to place point C in the segment AB so that

$$\frac{AC}{AB} = \frac{BC}{AC} \tag{6}$$

Golden Section Algorithm

1. Start with a bracketing triplet $x^0 < x^2 < x^1$ so that $f(x^2) < f(x^1), f(x^0)$ and $x^2 - x^0 = R(x^1 - x^0)$.
2. Choose x^3 so as to divide the longest of x^0x^2, x^1x^2 in two parts having ratio R
3. If $f(x^3) < f(x^2)$

   then $x^1 \leftarrow x^2$, $x^2 \leftarrow x^3$ (this eliminates an interval representing a fraction $1 - R$ of the initial interval)
   else $x^0 \leftarrow x^3$ (this eliminates an interval representing the fraction $R \times R = 1 - R$ of the initial interval)
4. Stop if $x^1 - x^0 < tol$ a desired tolerance, else repeat the previous step

Note that after each step, the interval containing the minimum shrinks by $R = 0.681\ldots$. The value R is optimally chosen so that the reduction in step 3 of the algorithm is the same in either case. If the initial bracketing triplet is not in the golden ratio R, it can be shown that the subsequent ratios will converge towards R.

How small shall we make the tolerance *tol* be? Near the (unidimensional) minimum b, $f(x) \approx f(b) + (x-b)^2/2f''(b)$ with the second term being much smaller than the first. Hence, when this term becomes a factor of ϵ smaller, it will be negligible when added to the first, and we may just as well stop the iteration. This happens when

$$|x - b| < \sqrt{\epsilon}|b|\sqrt{\frac{2f(b)}{f''(b)b^2}} \tag{7}$$

The last $\surd$ is often of order 1, therefore the above formula implies that the tolerance should be set to be of order $\sqrt{\epsilon}|b|$ with ϵ being the ϵ-machine of the current implementation. For standard double precision numbers, $\sqrt{\epsilon} \approx 3.10^{-8}$.

If the line minimization is performed without first bracketing the minimum, i.e we look for the best $\alpha^k \in [0, s]$ for some arbitrarily chosen s, the method is called **truncated minimization**.

3.2 The Parabolic Interpolation method

If the function f is smooth then a method that assumes that will converge faster than the worst-case-safe Golden ratio rule. This is the parabolic interpolation method that assumes that the function is approximately a parabola in the interval considered.

Parabolic Approximation Algorithm

1. Start with a bracketing triplet $x^0 < x^2 < x^1$ so that $f(x^2) < f(x^1), f(x^0)$.
2. Fit a parabola through the three points and let x^3, $f(x^3)$ be its minimum. (We assume $f(x^3) < f(x^1)$, otherwise this method is not useful.)
3. If $x^3 \in (x^0, x^2)$

   then $x^2 \leftarrow x^3$, $x^1 \leftarrow x^2$
   else $x^0 \leftarrow x^2$
4. Stop if $x^1 - x^0 < tol$ a desired tolerance, else repeat the previous step

In NR you can find the **Brent algorithm** which combines the golden section and the parabolic interpolation algorithms. It attempts to use parabolic interpolation, but detects when this method fails to approach the minimum and switches to golden section.

3.3 The Armijo (Backtracking) Rule

Intuition Assume that we found a bracketing interval $[0,\ s]$ for α^k. We start with $\alpha^k = s$ and decrease it exponentially until we find that the function f

has decreased "enough". What is "enough"? In an infinitesimal interval near x^k along the direction of descent, the function will decrease linearly, hence

$$f(x^k) - f(x^k + \alpha d^k) \approx \alpha(-\nabla f(x^k)^T d^k) \tag{8}$$

For a finite interval, we will ask for a decrease in f that is at least $\sigma < 1$ smaller than the above. Note that for a sufficiently small α such a decrease can always be attained.

Armijo Line Search

1. Start with $\alpha^k = s$, $\beta < 1$, $\sigma < 1$
2. If $f(x^k) - f(x^k + \alpha^k d^k) > \sigma\alpha^k(-\nabla f(x^k)^T d^k)$
   - then STOP
   - else $\alpha^k \longleftarrow \beta\alpha^k$ and repeat

In practice, $\beta \approx 0.5$ and $\sigma << 1$ e.g 0.1, 0.01 or even 0; $s = 1$ if no bracketing is done.

3.4 The Wolfe conditions

These conditions are stronger than the condition ensured by the Armijo rule; they are useful to get optimal performance from the Conjugate Gradient and Quasi-Newton methods.

$$f(x^k) - f(x^k + \alpha^k d^k) > c_1\alpha^k(-\nabla f(x^k)^T d^k) \quad \text{(sufficient descent)} \tag{9}$$

$$\nabla f(x^k + \alpha^k d^k)^T d^k \geq c_2 \nabla f(x^k)^T d^k \quad \text{(curvature)} \tag{10}$$

$$\text{with} \quad 0 < c_1 < c_2 < 1 \tag{11}$$

The first condition is identical to the Armijo rule. The second one ensures that the step taken is not too small, by enforcing that the derivative along the search direction has increased relative to the derivative at 0. The constant c_2 is typically chosen near 1, .e.g $c_2 = 0.9$.

An algorithm to ensure the Wolfe conditions is given in NW (Algorithm 3.5, pages 60–61). The algorithm searches by sometimes decreasing the interval (to ensure sufficient decrease) and sometime enlarging it (to ensure curvature). Such an algorithm is usually part of the implementation of Quasi-Newton methods.

4 Multidimensional minimization. The choice of direction

4.1 The steepest descent method

The steepest descent method follows the direction of the gradient. It can be shown [B] that gradient descent with line minimization has a linear rate of convergence. For other line search methods, including constant step size, the rate of convergence is no larger.

The convergence coefficient β of equation (4) can get very close to 1 (very slow convergence) if the Hessian is ill conditioned. Let M, m denote respectively the largest and the smallest eigenvalue of $\nabla^2 f(x^*)$. By continuity, we can assume that the Hessian around x^* is approximately the same. If $M >> m$ then the function will have a "long, narrow valley" with an almost flat "bottom" around x^*, oriented along the smallest eigenvector. The gradient will be almost perpendicular to the valley, and the algorithm, even with the optimal line minimization, will advance very slowly. See also Part II for a more precise evaluation of this effect.

Hence, all the following methods (except for stochastic gradient) can be seen as "applying some coordinate transformation" that will turn the elongated ellipses into circles, so that steepest descent in this new coordinate frame can move rapidly towards the optimum. Equivalently, having such a transformation (which is represented by the Hessian matrix), one can apply the "inverse transformation" to the descent direction, which is precisely what the Newton-Raphson method does.

4.2 The Newton-Raphson method

Assume that our function is quadratic, i.e

$$f(x) = \frac{1}{2} x^T A x + b^T x + c \text{ with } A \succ 0. \tag{12}$$

Then,

$$\nabla f(x) = Ax + b \tag{13}$$
$$\nabla^2 f(x) = A \tag{14}$$

and the minimum can be computed analytically as the solution of $Ax+b = 0$, namely $x^* = -A^{-1}b$. Equivalently, for any x

$$x^* - x = -A^{-1}b - A^{-1}Ax = -A^{-1}(Ax + b) = -\nabla^2 f(x)^{-1}\nabla f(x) \tag{15}$$

Hence, if f is quadratic, from any point x we can move in one step equal to $-\nabla^2 f(x)^{-1}\nabla f(x)$ to the minimum. Therefore, the Newton-Raphson method takes $D^k = \nabla^2 f(x)^{-1}$ as if the function was quadratic. Usually one also does a line search method, i.e $\alpha^k \neq 1$ in practice.

Newton-Raphson is practically and theoretically very fast once we are in the vicinity of the optimum (Part II of this lecture gives one result). However, its behavior far away from the optimum must be monitored carefully. Note for example that this is not a descent method, in the sense that it's not guaranteed that $f(x^{k+1}) < f(x^k)$ unless some form of line minimization is used. Also, the method is attracted by local maxima just as much as by local minima, so attention must be paid any time the Hessian is not positive definite. See [B] for modern methods that deals with these problem (trust region method and variations of quasi-Newton methods).

Another drawback is the need to compute and store the Hessian ($\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^{2--3})$ operations). Computation makes Newton-Raphson prohibitive in high dimesions.

A quick fix called **diagonal scaling**, where only the diagonal terms of the Hessian are computed.

$$D^k = \text{diag}\left\{\frac{\partial^2 f}{\partial x_i^2}\right\} \tag{16}$$

This method is obviously linear in storage and number of operations but it tends to underestimate the ratio M/m. Diagonal scaling amounts to rescaling each variable separately, and it is effective in those cases when the variables have very different ranges because of "imbalanced" measurement units (e.g in one direction the unit is miles, in the other one it is millimeters).

It is useful to make also the general observation that the Newton method is "scale free", i.e it is unaffected by linear coordinate changes.

Computing the direction Let $H = \nabla^2 f(x^k)$, $g = \nabla f(x^k)$. Computing the descent direction $d = -H^{-1}g$ by inverting the Hessian is neither the most efficient, nor the most exact method in terms of numerical error. Here are two alternatives:

1. Solve the system $Hd = -g$. The matrix H being symmetric, $\succ 0$, it can be factored into a product $H = LL^T$, with L lower triangular. This is called the **Cholesky factorization** and takes $\sim \frac{n^3}{6}$ operations. Solving an upper/lower triangular linear system is $\sim \frac{n^2}{2}$

2. Minimize the quadratic function $\frac{1}{2}d^T Hd + g^T d$, whose gradient equals $Hd + g$. Of course, the minimization should be done by a method other than Newton-Raphson! A possibility is to use steepest descent, starting from $d = -g$. Note that after every step, the current approximation d of the solution is a descent direction. The number of flops per iteration for this method is $\sim \frac{n^2}{2}$.