Numerical Optimization Ch.2

This is a note doc of *Numerical Optimization*.

Fundamentals of Unconstrained Optimization

Basic Knowledge

Some Definition

> A point x^* is a **global minimizer** if $f(x^*) \leq f(x)$ for all x.

> A point x^* is a **local minimizer** if there is a neighborhood $\mathcal{N}$ s.t. $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$.

> A point x^* is a **local minimizer** if there is a neighborhood $\mathcal{N}$ s.t. $f(x^*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x^*$.

First-Order Necessary Theorm

> If x^* is a local minnimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.

We call x^* a **stationary point** if $\nabla f(x^*) = 0$.

Second-Order Necessary Theorm

> If x^* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open nerghborhood of x^*, and then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

Second-Order Sufficient Conditions

> Suppose that $\nabla^2 f$ is continuous in an open nerghborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.

Thm.2.5 Sufficient Conditions for Convex Function

> When f is **convex**, any local minimizer x^* is a global minimizer of f.
> If f is both **convex** and **differentiable**, then any stationary x^* is a global minimizer of f.

f is convex $\Leftrightarrow$

$$\forall x_1, x_2 \in X, \forall t \in [0,1]: \ f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)$$

f is strictly convex $\Leftrightarrow$

$$\forall x_1, x_2 \in X, \forall t \in [0,1]: \ f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$$

Overview of Algorithms

All algorithm for unconstrained minimization *require* the user to supply a **starting point**, which we usually deenote by x_0.

> The user with knowledge about the application and the data set may be in a good position to choose x_0 to be a reasonable estimate of the solution.

> Otherwise, the starting point must be chosen by the algorithm, either by a systematic approach or in some arbitrary manner.

Begining at x_0, the algorithms generate a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ that terminate when either get enough accuracy or no more progerss can be made. The algorithm iterates x_k to x_{k+1} with the information about the function f at x_k, and possibly also information from earlier iterates $x_0, x_1, \ldots, x_{k-1}$.

Two Fundanmeental Strategies

1. **Line Search**
   In this strategy, the algorithm choose a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower funciton value. The main idea of every iteration is:

$$\min_{\alpha>0} f(x_k + \alpha p_k)$$

However, we do not always solve it exactly cause it may be expensive.

2. **Trust Region**
   In this strategy, we use the inforamtion gathered about f to construct a *model function* m_k whose behavior near x_k is similar to f. The main subproblem in every iteration is:

$$\min_{p \text{ near } x_k} m_k(x_k + p)$$

   A usually defined model is the quadratic function:

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p$$

   where the matrix B_k is either the Hessian $\nabla^2 f_k$ or some other approximation.

In a sense, this two strategies differ in the order in which they choose the *direction* and *distance* of the move to the next iterate. Line search starts by fixing the direction p_k while trust region restricts a maximum distance first.

Search Direction for Line Search Methods

1. **The steepest descent direction --∇f_k**
   most obvious choice for search direction
2. **Newton direction**
   perhaps the most important one of all
   The direction is derived from the second-order Taylor series approximation to $f(x_k + p)$, which is

$$f(x_k + p) \approx f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p \overset{def}{=} m_k(p)$$

   Assuming for the moment that $\nabla^2 f_k$ is positive deefinite, we obtain the Newton direction by minimizes $m_k(p)$, i.e. setting the derivative of $m_k(p)$ to zero, and we get:

$$p_k^N = -(\nabla^2 f_k) \nabla f_k$$

   Note that this direction is reliable only when the approximation is accurate.
   Newton direction can be used when $\nabla^2 f_k$ is positive definite, for in this case we have

$$\nabla f_k^T p_k^N = -{p_k^N}^T \nabla^2 f_k p_k^N \le -\sigma_k ||p_k^N||^2$$

for some $\sigma_k > 0$.
Unlike the steepest descent direction, there is a "natural" step length of 1 associated with the Newton direciotn.
When $\nabla^2 f_k$ is not positive definite, the Newton directoin may not even be defined.
The **main drawback** of Newton dirction is the neeed for the Hessian $\nabla^2 f(x)$, cause the process can sometimes be cumbersome, error-prone and expensive.

3. **Quasi-Newton search directions**
It provides an alternative to Newton's method in that they do not require the Hessian without lossing the convergence. In place of the true Hessian, they use an approxiamtion B_k, which is updated after each step to take account of the additoinal knowledge gained during the step.

$$\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 [\nabla^2 f(x+tp) - \nabla^2 f(x)]p \, \mathrm{dt}$$

By setting $x = x_k, p = x_{k+1} - x_k$, we obtain

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f(x)(x_{k+1} - x_k) + o(||x_{k+1} - x_k||)$$

Then we get the approxiamtion

$$\nabla^2 f_k(x_{k+1} - x_k) \approx \nabla f_{k+1} - \nabla f_k$$

Then we get the following method:

$$B_{k+1} s_k = y_k$$

where

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k$$

Typically, we impose additional conditions on B_{k+1}, such as symmetry and a requirement that the differenc between succssive approximations B_k and B_{k+1} have low rank.
3.1. *symmetric-rank-one(SR1) formula*

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

3.2 *BFGS formula*

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

The Quasi-Newton search direction is obtained by using B_k in place of the exact Hessian, that is,

$$p_k = -B_k^{-1} \nabla f_k$$

Note that we use the inverse of B_k instead of B_k itself. In fact, the equivalent formula for the former equations, applied to the inverse approximation $H_k \overset{def}{=} B_k^{-1}$, is

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \ \rho_k = \frac{1}{y_k^T s_k}$$

Then $p_k = -H_k \nabla f_k$

4. **Nonlinear Conjugate Gradient Method(NCG)**
They have the form

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1}$$

where β_k is a scalar that ensurees p_k and p_{k-1} are *conjugate*--- which is an important concept in the future.
CG methods were originally designed to solv systems of linear eqautions $Ax = b$, where A is symmetric and positive definite. *The problem of solving this linear system is equivalent to the problem minimizing the convex quadratic function defined by*

$$\phi(x) = \frac{1}{2} x^T A x - b^T x$$

In general, NCG are much mor effective than the steepest deescent direction and are almost as simplee to compute. These methods do not attain the fast convergence rates of Newton or quasi-Newton methods. But they have advantages of not requiring storage of matricees.

Models for Trust-Region Methods

1. Set $B_k = 0$, define trust region with Euclidan norm
we get

$$\min_p f_k + p^T \nabla f_k \quad \text{s.t. } ||p||_2 \leq \Delta_k$$

$$\Rightarrow p_k = -\frac{\Delta_k \nabla f_k}{||\nabla f_k||}$$

2. Set B_k to be the exact Hessian
   Because $||p||_2 \leq \Delta_k$, the subproblem is guaranteed to have a solution. And this method is highly effective in practice.

Scaling

In unconstrained optimization, a problem is said to be *poorly scaled* if changes to x in a certain direction produce much larger variations in the value of f than do changes to x in another direction, for example, $f(x) = 10^9 x_1^2 + x_2^2$ is very sensitive to small changes in x_1, but not so sensitive to perturbations in x_2.

Some optimization algorithms, such as steepest descent, are sensitive to poor scaling, while others, such as Newton's method, are unaffected by it.

Algorithms that are not sensitive to scaling are preferable, because they can handle poor problem formulations in a more robust fashion. And generally speaking, it is easier to preserve scale invariance for line search algorithms than for trust-region algorithms.