**EE 381V: Large Scale Optimization** **Fall 2012**

# Lecture 10 — October 2

*Lecturer: Caramanis & Sanghavi* *Scribe: Rajiv Khanna & Subhashini Krishnasamy*

Reference: Nocedal & Wright - Numerical Optimization, Ch 6. in the new edition, Ch. 8 in the original.

## 10.1 Last time

In the last lecture, we talked about methods of scaling up Newton's method, without compromising too much on its faster convergence guarantees. We studied Conjugate Gradient method that solves linear systems efficiently. We then extended it to a Non-linear Conjugate Gradient method which is more generic. We briefly touched upon another class of methods called Quasi-newton methods that approximate the Hessian in an iterative manner to bypass its computation and inversion done by Newton's method. In this lecture, we look at Quasi-newton methods in more depth.

## 10.2 Quasi-newton methods

While the convergence guarantees of Newton's method allow for quadratic rates in the number of iterations it requires, each individual iteration is computationally intensive - of the order of $O(n^3)$ - as it requires computation of the Hessian and then its inverse. Finding a way around computing the Hessian and its inverse motivates a new set of alternate methods called Quasi-newton methods.

Recall that the Newton method optimizes over the quadratic approximation of the original function $f$:

$$\hat{f}_k(x_k + \Delta x) = f(x_k) + \nabla f^T(x_k)\Delta x + \frac{1}{2}\Delta x^T(\nabla^2 f(x_k))\Delta x. \tag{10.1}$$

Quasi-Newton methods replace the Hessian in Equation [10.1] by another positive definite matrix $B_k$ that is intended to approximate the Hessian. These methods obtain speed up over Newton's method as follows - instead of computing $B_k$ afresh at every iteration, it is updated by application of a simpler modification (than inverting the Hessian) to factor in the curvature information obtained at the latest step. Hence, the curvature information would play an important role in designing $B_k$.

### 10.2.1 Designing $B_k$

Consider an iterative Newton-like optimization that uses $B_k$ instead of the Hessian. At steps $k$ and $k+1$, we have the quadratic approximation $\hat{f}$ of the function $f$ as shown in Equations

[10.2],[10.3].

$$\hat{f}_k(x_k + \Delta x) = f(x_k) + \nabla f^T(x_k)\Delta x + \frac{1}{2}\Delta x^T B_k \Delta x. \tag{10.2}$$

$$\hat{f}_{k+1}(x_{k+1} + \Delta x) = f(x_{k+1}) + \nabla f^T(x_{k+1})\Delta x + \frac{1}{2}\Delta x^T B_{k+1} \Delta x. \tag{10.3}$$

To generate the next iterate $x_{k+1}$, we have the quasi-newton step as:

$$x_{k+1} = x_k - \eta_k B_k^{-1} \nabla f(x_k). \tag{10.4}$$

There may be several methods of choosing feasible $B_{k+1}$. One reasonable way to pick $B_{k+1}$ is by ensuring that the gradient of $\hat{f}_{k+1}$ at the iterates $x_k$ and $x_{k+1}$ matches with the gradient of $f$ at those points as show in Equations [10.5],[10.6 respectively

$$\nabla \hat{f}_{k+1}(x_k) = \nabla f(x_k). \tag{10.5}$$

$$\nabla \hat{f}_{k+1}(x_{k+1}) = \nabla f(x_{k+1}). \tag{10.6}$$

If we take the derivative of Equation [10.3] with respect to $\Delta x$, at $\Delta x = 0$, we see that Equation [10.6] holds automatically.

We should pick $B_{k+1}$ so that Equation [10.5] also holds. This is done by taking the derivative of Equation [10.3] with respect to $\Delta x$, at $\Delta x = x_k - x_{k+1}$ and setting thus obtained derivative to $\nabla f(x_k)$, to get:

$$B_{k+1} s_k = y_k, \tag{10.7}$$

where,

$$s_k = x_{k+1} - x_k,$$

and

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

Equation [10.7] is also called the **Secant equation**.

**Exercise 1** Check that the Secant Equation has a positive definite solution for $B_{k+1}$ iff $y_k^T s_k > 0$.

*Solution:* If we restrict the choice of $B_{k+1}$ to positive definite matrices, from Secant equation, we get $s_k^T B_{k+1} s_k = s_k^T y_k > 0$. Conversely, if $s_k^T y_k > 0$, we can always find a positive semidefinite solution $B_{k+1}$ to the Secant equation. To see this, we note that there are enough degrees of freedom to achieve the goal of a psd $B_{k+1}$. In addition to the $(n^2 - n)/2$ conditions imposed by the symmetry of the matrix, $B_{k+1}$ has to satisfy the $n$ inequalities imposed by the positive definiteness and also the $n$ conditions imposed by the Secant equation. This still leaves us with $n(n-3)/2$ degrees of freedom for the $nXn$ matrix. Thus, there are infinitely many solutions for problems of dimension greater than three.

**Exercise 2** Check that if $f(x)$ is convex, it is always true that $y_k^T s_k > 0$.

*Solution:* When $f$ is strictly convex, $s_k^T y_k > 0$ for any points $x_k$ and $x_{k+1}$. This can be proved by using the first order condition for convexity.

$$\begin{aligned}
f(x_{k+1}) &> f(x_k) + \nabla f(x_k)^T(x_{k+1} - x_k) \\
&> f(x_{k+1}) + \nabla f(x_{k+1})^T(x_k - x_{k+1}) + \nabla f(x_k)^T(x_{k+1} - x_k) \\
\implies \quad 0 &> (\nabla f(x_{k+1})^T - \nabla f(x_k)^T)(x_k - x_{k+1}) \\
\implies \quad s_k^T y_k &> 0
\end{aligned}$$

### 10.2.2 DFP

To constrain our choice of $B_{k+1}$, we choose one that is "closest" to $B_k$ in some sense.

$$\begin{aligned}
B_{k+1} = \quad & \operatorname{argmin}_B ||B - B_k||_W \\
\text{s.t.} \quad & Bs_k = y_k, \\
& B \succ 0.
\end{aligned} \tag{10.8}$$

In the Equation [10.8], $||.||_W$ is the Weighted Frobenius Norm using a matrix $W$. Note that we get the same solution $B_{k+1}$ for various choices of $W$.

**Aside:**

- *Frobenius norm:* The Frobenius norm of a matrix $A$ is defined as

$$||A||_F \triangleq (\sum_{i,j} A_{i,j}^2)^{\frac{1}{2}}.$$

- *Weighted Frobenius norm:* The Weighted Frobenius norm of a matrix $A$ w.r.t another matrix $W(W \succ 0)$ is defined as

$$||A||_W \triangleq ||W^{\frac{1}{2}} A W^{\frac{1}{2}}||_F.$$

- *Spectral theorem:* Recall that if $W$ is symmetric, it can be written as $W = \sum_i \lambda_i x_i x_i^T$, where $\lambda_i$ is the eigenvalue associated with the eigenvector $x_i$. All $x_i$ are orthonormal to each other.

- For a psd $W$, $W^{\frac{1}{2}}$ exists, and can be written as $W^{\frac{1}{2}} = \sum_i \lambda_i^{\frac{1}{2}} x_i x_i^T$.

- *Sherman-Morrison-Woodbury formula* : If a matrix undergoes a rank $k$ update as

$$A_+ = A + UV^T,$$

where $U$,$V$ are rank $k$ matrices, then $A^{-1}$ follows the update rule:

$$A_+^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1} U)^{-1} V^T A^{-1}. \tag{10.9}$$

Equation [10.9] is also called *Matrix Inversion Lemma*, and can be easily verified by multiplying definitions of $A_+$ and $A_+^{-1}$ to get the identity matrix.

**Lemma:** If we choose $W$ to be the average Hessian, defined as:

$$W = \int_0^1 \nabla^2 f(x_k + t\eta_k \Delta x_k)dt,$$

then Equation [10.8] has a unique solution given by

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

where

$$\rho_k = \frac{1}{y_k^T s_k}.$$

The above lemma gives us $B_{k+1}$ which is an approximation of the Hessian, not its inverse. To use it in the quasi-newton step (Equation [10.4]), we need to compute its inverse. Finding $B_k$ at every iteration $k$ and then inverting it is costly, but by using Sherman-Morrison-Woodbury formula (Equation [10.9]), we can update $H_k = (B_k)^{-1}$ directly as

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}. \tag{10.10}$$

The update method of Equation [10.10] represents a quasi-newton method called **DFP**, after its inventors Davidon, Fletcher and Powell. Intuitively, the update step combines the newly found curvature information in the current step into the approximate Hessian matrix as a rank-two update, thus constraining the next iteration $H_{k+1}$ to not stray "too far" from the current estimate $H_k$.

### 10.2.3 BFGS

Different methods of updating $B_k$, $H_k$ yield different algorithms. Another update method, which works well practically is the **BFGS** method (named after its inventors - Broyden, Fletcher, Goldfarb, and Shanno). While DFP attempts to solve the secant equation given by Equation [10.7], BFGS solves an alternative secant equation that approximates the inverse of the Hessian directly:

$$s_k = H_{k+1} y_k, \tag{10.11}$$

by

$$\begin{aligned} H_{k+1} = \ & \text{argmin}_H \, ||H - H_k||_W \\ \text{s.t.} \ & s_k = H y_k, \\ & H \succ 0. \end{aligned} \tag{10.12}$$

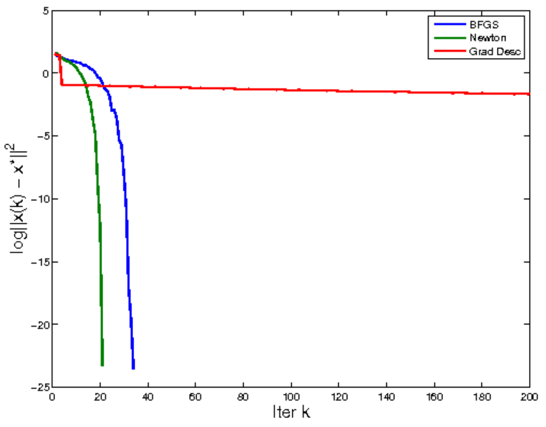

**Figure 10.1.** Convergence over Rosenbrock function (Equation [10.14])

The solution to Equation [10.12] is similar to that of Equation [10.8], with the roles of $s_k$ and $y_k$ interchanged.

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T,$$
$$\rho_k = \frac{1}{y_k^T s_k}. \tag{10.13}$$

We have the update equations. But how do we select $B_0$ or $H_0$? Unfortunately, there is no set formula for this. It depends on the problem at hand. Setting it to the identity matrix, or a multiple of it, works well for many problems. Note that if we set $H_0$ to the identity matrix, the first update is the same as gradient descent, while subsequent steps get more and more refined as $H_k$ approximates the Hessian better in subsequent steps.

**Theorem:** *Superlinear convergence of BFGS:* If $f$ is twice continuously differentiable, has an $L-$Lipschitz Hessian and is strongly convex for the level sets of $\forall x \leq x_0$, then $x_k \longrightarrow x^*$ superlinearly.

BFGS is widely accepted as the "best" practical Quasi-Newton method available. Its advantages over Newton method are clear - it does not solve a linear system that Newton's method does (by Conjugate Gradients or matrix inversion), and does not need to compute the second derivatives. Furthermore, in practice, the number of iterations it takes to converge are almost as few as Newton's method. For a comparison, see Figure [10.1] that illustrates convergence rates of Newton's method, BFGS and Gradient Descent over Rosenbrock function given by

$$f(x) = 100(x_0 - x_1^2)^2 + (1 - x_1)^2. \tag{10.14}$$

For quadratic functions, BFGS is guaranteed to terminate in $n$ steps, where $n$ is the dimension of the problem space. Also, in quadratic problems, $H_n$, the final approximate matrix, is actually the true Hessian. However, this is not true for non-quadratic problems - $H_k$ may not converge to the true Hessian in general.

```
**Data**: Dimension n, an oracle that can be queried to get $\nabla f(x)$ for any $x$, tolerance $\epsilon$
**Result**: $x* = \underset{x}{\operatorname{argmin}} f(x)$
Initialization: $H_0 = I_n, k = 0$, some starting point $x_0$;
while $||\nabla f|| > \epsilon$ do
    $\Delta x_k = -H_k \nabla f(x_k)$
    Quasi-Newton step (use line search to get $\eta_k$: $x_{k+1} = x_k + \eta_k \Delta x_k$
    $s_k = x_{k+1} - x_k$
    $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
    Update $H_{k+1}$ as in Equation [10.13]
    $k = k + 1$
end
```

**Algorithm 1:** BFGS

For large dimensional problems, computing and storing $H$ may be hard. This motivates a widely popular low-memory variant of BFGS called L-BFGS. Instead of storing and manipulating $H_k$ at each iteration, L-BFGS stores and manipulates the $y_k$ and $s_k$ from the last $m$ iterations, where $m << n$.

## 10.3 Constrained Optimization

In the last few lectures, we focused on algorithms for unconstrained optimization. We now discuss the analytical characterization of constrained optimization problems and then describe algorithms for solving these problems. Specifically, we are interested in convex optimization problems since, for these problems, a locally optimal point is also globally optimal. A generic convex problem can be written as $\min_{x \in \mathcal{X}} f(x)$, where $\mathcal{X}$ is a convex set and $f(\cdot)$ is a convex function.

Recall that we defined *tangent cone* of a point $x$ w.r.t a set $C$ as the closure of the set of all feasible directions from the point $x$ in the set $C$. Also, we defined *normal cone* as the polar cone of the tangent cone.

**Proposition**: Let $\mathcal{X}$ be convex and $x \in \mathcal{X}$. If $z \in N_{\mathcal{X}}(x)$, then $Proj_{\mathcal{X}}(z) = x$.

**Proof:** By the definition of the normal cone, if $z \in N_{\mathcal{X}}(x)$, then $\langle z - x, y - x \rangle \leq 0 \ \forall y \in \mathcal{X} \implies x = Proj_{\mathcal{X}}(z)$ (see variational characterization of projection in Lecture 3). $\square$

We also characterized the optimal point in terms of the normal cone: $x^*$ is optimal iff $0 \in \nabla f(x^*) + N_{\mathcal{X}}(x^*)$. For non-smooth functions, $\nabla f(x^*)$ can be replaced by the set of all straight lines passing through $x^*$ that lie below $f(x)$.

## 10.4 Duality for Linear Programming Problems

Consider the standard linear programming problem,

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & a_i^T x \geq b_i \end{array} \tag{10.15}$$

Figure 10.2 shows the vector $c$ and the polyhedron formed by the halfspaces $a_i^T x \geq b_i$. If

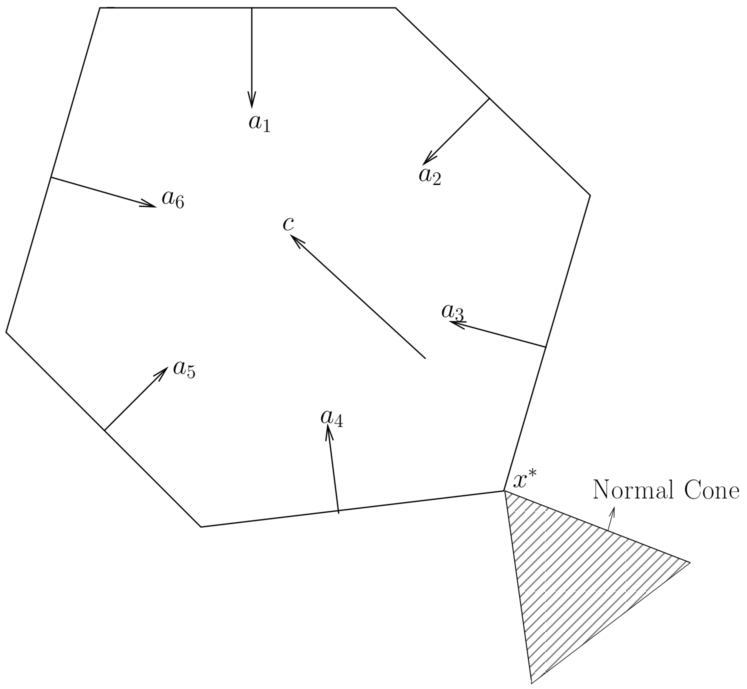

**Figure 10.2.** An example linear programming problem

$x^*$ is optimal, then $-c = -\nabla f(x^*) \in N_{\mathcal{X}}(x^*)$. Observe that

$$N_{\mathcal{X}}(x^*) = \{-(\lambda_3 a_3 + \lambda_4 a_4) \mid \lambda_3, \lambda_4 \geq 0\}$$

Therefore, $x^*$ is optimal iff $\exists\, \lambda_3^*, \lambda_4^*$ such that $c = \lambda_3^* a_3 + \lambda_4^* a_4$. Since $a_3^T x \geq b_3$ and $a_4^T x \geq b_4$,

$$\begin{array}{rcl} & (\lambda_3^* a_3 + \lambda_4^* a_4)^T x & \geq \lambda_3^* b_3 + \lambda_4^* b_4 \\ \implies & c^T x & \geq \lambda_3^* b_3 + \lambda_4^* b_4 \end{array} \tag{10.16}$$

$\lambda_3^*$, $\lambda_4^*$ are related to the solution of another programming problem given by

$$\begin{array}{ll} \max & \lambda^T b \\ \text{s.t.} & \sum_i \lambda_i a_i = c \\ & \lambda_i \geq 0 \end{array} \tag{10.17}$$

This is called the *dual* problem of the original (*primal*) optimization problem. The *Weak Duality Theorem* states that for any primal feasible $x$ and any dual feasible $\lambda$, $\lambda^T b \leq c^T x$.