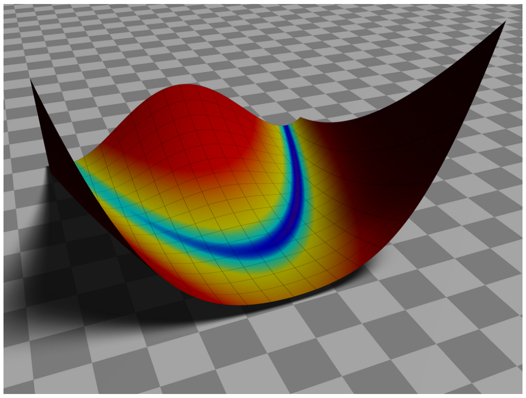
Unconstrained Optimization

Lecture 6



ME EN 575
Andrew Ning
aning@byu.edu

Outline

Secant Rule

BFGS

Newton's method: requires Hessian (this is a problem for us)

Quasi-Newton: estimate it.

New problem: how to estimate it?

How many degrees of freedom does a Hessian have?

Secant Rule

Fundamental idea: Instead of starting over on a Hessian estimate every iteration, let's just update it. We will update it by accounting for the curvature (change in gradient) during the most recent step.

Secant rule

1D analogue:

$$f_k + f'(x)(x_{k+1} - x_k) = f_{k+1}$$

Apply to derivative:

$$g_k + f''(x)(x_{k+1} - x_k) = g_{k+1}$$

Extend to nD:

$$g_k + H(x_{k+1} - x_k) = g_{k+1}$$

Rearange:

$$\boxed{H_{k+1} s_k = y_k} \qquad (s_k = x_{k+1} - x_k,\ y_k = g_{k+1} - g_k)$$

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

Always has a solution, *but* ...

- How many degrees of freedom?
- How many constraint equations?

More constraints are needed.

Additional constraint: of all the possible approximate Hessians to choose, let's pick the closest one to our current iterate.

BFGS

Fundamental Idea #2: What we really care about is solving:

$$p_k = -H_k^{-1} g_k$$

So, let's not bother estimating the Hessian. Instead, let's estimate the inverse of the Hessian!

$$p_k = -V_k g_k$$

- design variables: estimate V
- Constraint: symmetric
- Constraint: positive definite
- Constraint: secant rule $Hs = y \Rightarrow s = Vy$
- Objective: of all the possibilities, let's choose to closest one to last iteration.

We can solve this with optimization.

$$
\begin{aligned}
\text{minimize} \quad & \|V - V_k\| \\
\text{with respect to} \quad & V \\
\text{subject to} \quad & V = V^T \\
& V y_k = s_k
\end{aligned}
$$

Solution:

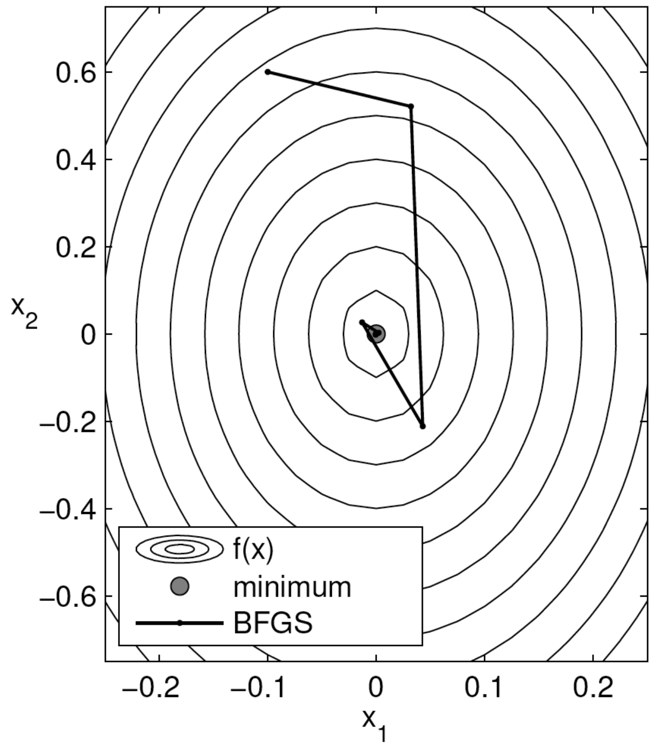
$$V_{k+1} = \left[I - \frac{s_k y_k^T}{s_k^T y_k}\right] V_k \left[I - \frac{y_k s_k^T}{s_k^T y_k}\right] + \frac{s_k s_k^T}{s_k^T y_k}.$$

But how do we ensure this is positive definite?

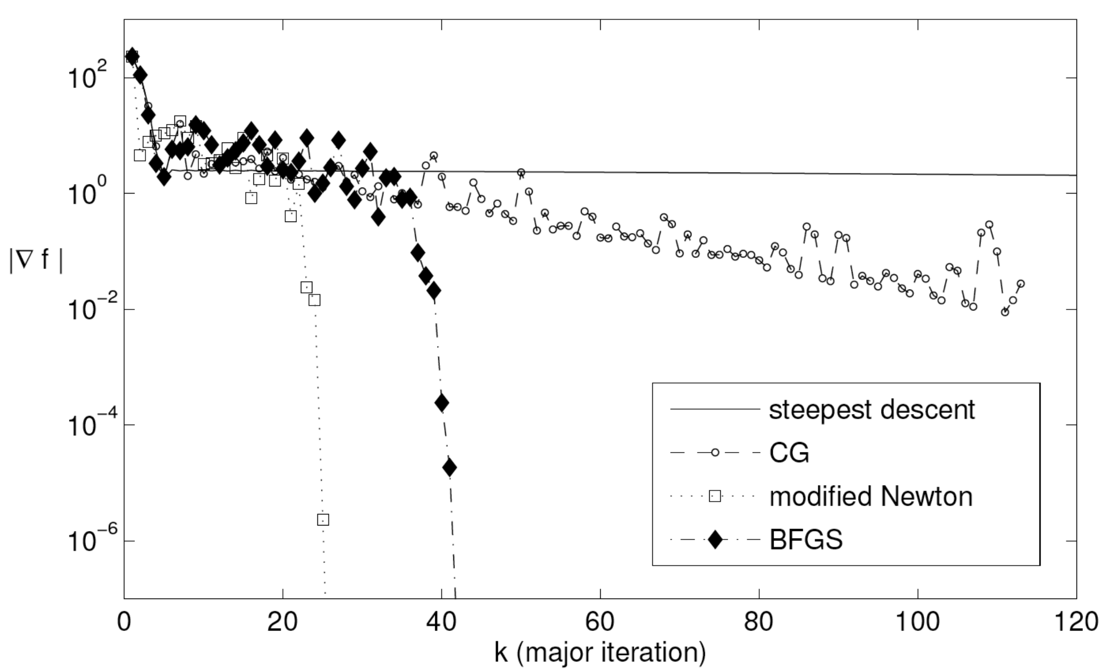
With this algorithm it is automatic, as long as previous estimate is postive definite.

What are some good options for a starting Hessian estimate?

Using the identity matrix in the first step corresponds to which method?


0.6
0.4
0.2
0
−0.2
−0.4
−0.6
x2
−0.2
−0.1
0
0.1
0.2
x1
f(x)
minimum
BFGS

Rosenbrock


10^2
10^0
10^−2
10^−4
10^−6
|∇ f |
0
20
40
60
80
100
120
k (major iteration)
steepest descent
CG
modified Newton
BFGS

Is fewer iterations always better?

Newton's method takes fewer iterations, but each iteration is more computationally expensive.

2 reasons:

- Requires second derivatives.
- Requires solving a linear system vs a matrix vector multiply. ($\mathcal{O}(n^3)$ vs $\mathcal{O}(n^2)$)

Review

Steepest Descent: $$p_k = \frac{-g_k}{\|g(x_k)\|}$$

Conjugate Gradient: $$p_k = \frac{-g_k}{\|g(x_k)\|} + \beta_k p_{k-1}$$

Newton: $$p_k = -H^{-1} g_k$$

Quasi-Newton: $$p_k = -V_k g_k$$

Matlab example using fminunc

Python example using minimize

Trust Region Methods

Current methods: pick search direction, then pick step along that direction.

Trust region: pick a "step" first, then choose a direction. Also, when "backtracking", the direction can change.