Newton's Method and BFGS

Let us again, consider the unconstrained, smooth convex optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$. Here f is convex, differentiable and defined over the full $\mathbb{R}^n$.

Recall that gradient descent chooses an initial value $\mathbf{x}^{(0)} \in \mathbb{R}^n$, and repeats $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - t_k \bigtriangledown f(\mathbf{x}^{(k-1)})$. In comparison, Newton's method repeats

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - (\bigtriangledown^2 f(\mathbf{x}^{(k-1)}))^{-1} \bigtriangledown f(\mathbf{x}^{(k-1)})$$

Here $\bigtriangledown^2 f(\mathbf{x}^{(k-1)})$ is the Hessian matrix of f at $\mathbf{x}^{(k-1)}$.

1 Newton's method interpretation

Instead of using the quadratic approximation term from gradient descent, we can use the exact Taylor series expansion for $f(\mathbf{y}) \approx f(\mathbf{x}) + \bigtriangledown f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{1}{2}(\mathbf{y}-\mathbf{x})^T \bigtriangledown^2 f(\mathbf{x})(\mathbf{y}-\mathbf{x})$. In order to minimize this expression, we differentiate RHS with respect to $\mathbf{y}$ and equate it to zero. We get,

$$\bigtriangledown f(\mathbf{x}) + \bigtriangledown^2 f(\mathbf{x})(\mathbf{y}-\mathbf{x}) = 0 \implies \mathbf{y} = \mathbf{x} - (\bigtriangledown^2 f(\mathbf{x}))^{-1} \bigtriangledown f(\mathbf{x}),$$

and thus the update for Newton's method.

In case of minimization of $f(\mathbf{x})$, we essentially solve for $\bigtriangledown f(\mathbf{x}) = 0$. History: work of Newton (1685) and Raphson (1690) originally focused on finding roots of polynomials. Simpson (1740) applied this idea to general nonlinear equations, and minimization by setting the gradient to zero. Alternative proof of above update focusing on solving $\bigtriangledown f(\mathbf{x}) = 0$ is given below. By Taylor expansion, we get

$$\bigtriangledown f(\mathbf{x}_2) \approx \bigtriangledown f(\mathbf{x}_1) + \bigtriangledown^2 f(\mathbf{x}_1)(\mathbf{x}_2 - \mathbf{x}_1).$$

Since, we want to get to the $\mathbf{x}_2$ such that it solves $\bigtriangledown f(\mathbf{x}) = 0$. Thus, we set $\bigtriangledown f(\mathbf{x}_2) = 0$ and then solve for $\mathbf{x}_2$ in $\bigtriangledown f(\mathbf{x}_1) + \bigtriangledown^2 f(\mathbf{x}_1)(\mathbf{x}_2 - \mathbf{x}_1) = 0$ which gives us $\mathbf{x}_2 = \mathbf{x}_1 - (\bigtriangledown^2 f(\mathbf{x}_1))^{-1} \bigtriangledown f(\mathbf{x}_1)$. This is also called the 'Newton-Raphson update'.

Specifically, we execute the following steps for the minimization problem $\min_{\mathbf{x}} f(\mathbf{x})$

using some initialization $\mathbf{x}^{(0)}$:

$$\begin{aligned}
\mathbf{x}^{(1)} =& \mathbf{x}^{(0)} - (\bigtriangledown^2 f(\mathbf{x}^{(0)}))^{-1} \bigtriangledown f(\mathbf{x}^{(0)}) \\
\mathbf{x}^{(2)} =& \mathbf{x}^{(1)} - (\bigtriangledown^2 f(\mathbf{x}^{(1)}))^{-1} \bigtriangledown f(\mathbf{x}^{(1)}) \\
& \vdots \\
\mathbf{x}^{(t)} =& \mathbf{x}^{(t-1)} - (\bigtriangledown^2 f(\mathbf{x}^{(t-1)}))^{-1} \bigtriangledown f(\mathbf{x}^{(t-1)}) \\
& \vdots
\end{aligned}$$

End with some stopping rule like gradient descent

The steps for the root finding problem $g(\mathbf{x}) = 0$ using some initialization $\mathbf{x}^{(0)}$:

$$\begin{aligned}
\mathbf{x}^{(1)} =& \mathbf{x}^{(0)} - (\bigtriangledown g(\mathbf{x}^{(0)}))^{-1} g(\mathbf{x}^{(0)}) \\
\mathbf{x}^{(2)} =& \mathbf{x}^{(1)} - (\bigtriangledown g(\mathbf{x}^{(1)}))^{-1} g(\mathbf{x}^{(1)}) \\
& \vdots \\
\mathbf{x}^{(t)} =& \mathbf{x}^{(t-1)} - (\bigtriangledown g(\mathbf{x}^{(t-1)}))^{-1} g(\mathbf{x}^{(t-1)}) \\
& \vdots
\end{aligned}$$

End with some stopping rule like gradient descent

The above two problems are inherently the same as for the first problem, we are essentially solving $f'(\mathbf{x}) = 0$.

2 Newton Decrement and stopping rule

To measure proximity of $\mathbf{x}$ to $\mathbf{x}^*$ (the optimal value), we define the newton decrement as $\lambda(\mathbf{x}) = \sqrt{\bigtriangledown f(\mathbf{x})^T (\bigtriangledown^2 f(\mathbf{x}))^{-1} \bigtriangledown f(\mathbf{x})}$. This quantity naturally comes up when we think about the difference between $f(\mathbf{x})$ and the minimum of the quadratic approximation, which is $f(\mathbf{x}) + \bigtriangledown f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{1}{2}(\mathbf{y} - \mathbf{x})^T \bigtriangledown^2 f(\mathbf{x})(\mathbf{y} - \mathbf{x})$.

Specifically, $f(\mathbf{x}) - \min_{\mathbf{y}} f(\mathbf{x}) + \bigtriangledown f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{1}{2}(\mathbf{y} - \mathbf{x})^T \bigtriangledown^2 f(\mathbf{x})(\mathbf{y} - \mathbf{x}) = f(\mathbf{x}) - (f(\mathbf{x}) - \frac{1}{2} \bigtriangledown f(\mathbf{x})^T (\bigtriangledown^2 f(\mathbf{x}))^{-1} \bigtriangledown f(\mathbf{x})) = \frac{1}{2}\lambda(\mathbf{x})^2$ (Verify!!)

Hence, we can think of $\frac{1}{2}\lambda(\mathbf{x})^2 \leq \epsilon$ as a reasonable stopping condition.

3 Backtracking Line Search

One big difference between gradient descent and Newton's method is the absence of any step-length parameter. It could have been advantageous, however pure Newton's method usually faces convergence issues. This is because pure newton's method is not guaranteed to be a descent method. Hence, we use **damped Newton's method** and the update looks a bit similar to Gradient descent.

Specifically, the update would become, $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - t(\bigtriangledown^2 f(\mathbf{x}^{(k-1)}))^{-1} \bigtriangledown f(\mathbf{x}^{(k-1)})$. The step-length parameter is again chosen by applying backtracking line search.

We use parameters $\alpha \in (0, \frac{1}{2}]$ and $\beta \in (0, 1)$. At each iteration, we first start with $t = 1$ and run an inner loop : while $f(\mathbf{x} + t\mathbf{v}) > f(\mathbf{x}) + \alpha t \bigtriangledown f(\mathbf{x})^T \mathbf{v}$, we set $t = \beta t$ and $\mathbf{x}_{new} = \mathbf{x}_{old} + t\mathbf{v}$ Here, $\mathbf{v} = -(\bigtriangledown^2 f(\mathbf{x}))^{-1} \bigtriangledown f(\mathbf{x})$ which means $\bigtriangledown f(\mathbf{x})^T \mathbf{v} = -\lambda^2(\mathbf{x})$. So loop condition is $f(\mathbf{x} + t\mathbf{v}) > f(\mathbf{x}) - \alpha t \lambda^2(\mathbf{x})$

Here, starting with $t = 1$ at the beginning of each inner loop seems principled, since that is the pure Newton's method. So in spirit, we try to take the full newton step. But if that does not give us a decrease in the criterion (objective), we shrink the step size by a multiplicative factor. Just like our GD implementation, we can set $\alpha = 0$ while testing our code first.

4 Comparison to 1st-order methods

Newton's method is more accurate than Gradient descent. However, it takes more memory and more number of floating point operations (flops) which is of order $O(n^3)$ for n-dimensional parameter. For the same task, gradient descent requires $O(n)$ operations. The big issue is to do with the inversion step of the Hessian matrix. When the Hessian matrix has a lot of structure, the inversion step becomes less costly if we can successfully use those.

5 Equality-constrained Newton's Method

One of the most popular optimization problem we often face is the following,

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ subject to } \mathbf{A}\mathbf{x} = \mathbf{b}.$$

To me, the best way would be to deduce the general solution space for $\mathbf{x}$ and use that solve the optimization problem.

The general solution space for $\mathbf{A}\mathbf{x} = \mathbf{b}$ is given by $\mathbf{A}_N\mathbf{y} + \mathbf{x}_0$, where $\mathbf{A}_N$ is the null space of $\mathbf{A}$ and $\mathbf{A}\mathbf{x}_0 = b$. Thus, $\mathbf{A}\mathbf{A}_N\mathbf{y} = 0$ for any $\mathbf{y}$. Thus now solve $\min_{\mathbf{y}} f(\mathbf{A}_N\mathbf{y} + \mathbf{x}_0)$.

6 Quasi-Newton methods

When Hessian is too expensive or singular, then we can use the quasi-Newton method that approximates $\bigtriangledown^2 f(\mathbf{x})$ with some $\mathbf{H} \succ 0$ (i.e. positive definite matrix). The updates are such that $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - (\mathbf{H})^{-1} \bigtriangledown f(\mathbf{x}^{(k-1)})$, where, we approximate the Hessian ($\mathbf{H}$) at each step cheaply both in terms of application and storage. Quasi-Newton converges fast (superlinear) but not as fast as Newton, as an estimate, every n steps of quasi-Newton method make the same progress as one Newton step.

If the Hessian does not involve any contribution from the parameter, we do not need to consider Quasi-Newton methods.

Let $x^{(0)}$ be the initial value and $\mathbf{H}^{(0)}$ positive definite. Repeat the following steps for $k = 1, 2, 3, \ldots$

1. Solve $\mathbf{H}^{(k-1)}\mathbf{s}^{(k-1)} = - \bigtriangledown f(\mathbf{x}^{(k-1)})$
2. Update $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \mathbf{s}^{(k-1)}$
3. Compute $\mathbf{H}^{(k)}$ from $\mathbf{H}^{(k-1)}$.

Different quasi-Newton methods implement Step 3 differently. As $\mathbf{H}^{(k-1)}$ already contains info about the Hessian, we can use suitable matrix update to form $\mathbf{H}^{(k)}$ from $\mathbf{H}^{(k-1)}$.

6.1 Secant method

Before we get into more detail on the multivariate $\mathbf{x}$, let's first consider a simpler case of univariate x. Specifically, we want to solve $\min_x f(x)$ where x is univariate. Then Newton-Raphson update at *t-th* stage is given by,

$$x_t = x_{t-1} - \frac{f'(x_{t-1})}{f''(x_{t-1})}.$$

In this case, an approximation for $f''(x_{t-1})$ can be obtained using finite-difference from the first principle of differentiation $f''(x_{t-1}) \approx \frac{f'(x_{t-1}) - f'(x_{t-2})}{x_{t-1} - x_{t-2}}$. This leads to so-called secant method which consist of the following updating scheme,

$$x_t = x_{t-1} - f'(x_{t-1}) \frac{x_{t-1} - x_{t-2}}{f'(x_{t-1}) - f'(x_{t-2})}.$$

For multivariate case: The above-mentioned secant method provides a way to approximate $\mathbf{H}^{(k)}$. This might look a bit odd at first.

If we take Taylor expansion of $\bigtriangledown f(\mathbf{x}^{(k)})$ around $\mathbf{x}^{(k-1)}$, we have

$$\bigtriangledown f(\mathbf{x}^{(k)}) = \bigtriangledown f(\mathbf{x}^{(k-1)}) + \mathbf{H}^{(k)}(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = \bigtriangledown f(\mathbf{x}^{(k-1)}) + \mathbf{H}^{(k)} s^{(k-1)},$$

due to our updating rule in Step (2) above. Thus, our required quasi-Newton condition is

$$\mathbf{H}^{(k)}s^{(k-1)} = \bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)}).$$

This is also called the 'secant equation'.

6.2 BFGS

There are several ways to approximately solve the above secant equation to get $\mathbf{H}^{(k)}$. We specifically see the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method. There are several Quasi-Newton methods that are built on the idea behind BFGS. New ideas are coming in as well.

BFGS algorithm assumes $\mathbf{H}^{(k)} = \mathbf{H}^{(k-1)} + a\mathbf{u}\mathbf{u}^T + b\mathbf{v}\mathbf{v}^T$, where $\mathbf{u}$ and $\mathbf{v}$ are vectors of length = number of rows or columns in $\mathbf{H}^{(k)}$. Note that $\mathbf{H}^{(k)}$'s are symmetric. So, this type of updating helps to main the symmetric structure.

What are the solutions of $a, b, \mathbf{u}$ and $\mathbf{v}$?

First, $\mathbf{H}^{(k)}s^{(k-1)} = \bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})$.

Thus, $\mathbf{H}^{(k-1)}s^{(k-1)} + a\mathbf{u}\mathbf{u}^T s^{(k-1)} + b\mathbf{v}\mathbf{v}^T s^{(k-1)} = \bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})$,

After some rearranging, $\{\bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})\} - \mathbf{H}^{(k-1)}s^{(k-1)} = a(\mathbf{u}^T s^{(k-1)})\mathbf{u} + b(\mathbf{v}^T s^{(k-1)})\mathbf{v}$.

There are 4 sets of unknowns and only one equation. Of course the solution will not be unique. BFGS sets the following sets of solutions.

$$\begin{aligned}
\mathbf{u} =& \{\bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})\} \\
a =& \frac{1}{(\mathbf{u}^T s^{(k-1)})} \\
\mathbf{v} =& \mathbf{H}^{(k-1)}s^{(k-1)} \\
b =& -\frac{1}{(\mathbf{v}^T s^{(k-1)})}.
\end{aligned}$$

Hence, $\mathbf{H}^{(k)} = \mathbf{H}^{(k-1)} + \frac{\{\bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})\}\{\bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})\}^T}{\{\bigtriangledown f(\mathbf{x}^{(k)}) - \bigtriangledown f(\mathbf{x}^{(k-1)})\}^T s^{(k-1)}} - \frac{\mathbf{H}^{(k-1)}s^{(k-1)}\{s^{(k-1)}\}^T\mathbf{H}^{(k-1)}}{\{s^{(k-1)}\}^T\mathbf{H}^{(k-1)}s^{(k-1)}}$

Using the Woodbury formula matrix inversion, we can compute $(\mathbf{H}^{(k)})^{-1}$ very cheaply. In fact, we get a formula to get $(\mathbf{H}^{(k)})^{-1}$ from $(\mathbf{H}^{(k-1)})^{-1}$ directly.

You can do this yourself. I will just give you the hint. The Woodbury matrix identity is

$$(\mathbf{A}+\mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1}+\mathbf{VA}^{-1}\mathbf{U}\right)^{-1}\mathbf{VA}^{-1},$$

Just set $\mathbf{A} = \mathbf{H}^{(k-1)}, \mathbf{U} = [\mathbf{u}; \mathbf{v}], \mathbf{C} = \text{diag}(a, b)$, and $\mathbf{V} = \mathbf{U}^T$. Then $\mathbf{H}^{(k)} = \mathbf{A} + \mathbf{UCV}$. Here $[\mathbf{u}; \mathbf{v}]$ means that the two column vectors are placed side-by-side to form a matrix with 2 columns.

Since the updates of the BFGS curvature matrix do not require matrix inversion, its computational complexity is only $\mathcal{O}(n^2)$, compared to $\mathcal{O}(n^3)$ in Newton's method.

Some variations: Symmetric rank-one update has also been considered where we let $\mathbf{H}^{(k)} = \mathbf{H}^{(k-1)} + a\mathbf{u}\mathbf{u}^T$. Here a and $\mathbf{u}$ are also solved in exact similar way.

Then there is Davidon-Fletcher-Powell update where it is assumed that $(\mathbf{H}^{(k)})^{-1} = (\mathbf{H}^{(k-1)})^{-1} + a\mathbf{u}\mathbf{u}^T + b\mathbf{v}\mathbf{v}^T$ and solved the parameters accordingly using the secant equation.

Even now, there are several groups working on different ways to cheaply compute the gradient or Hessian for big data with large sample size n and/or large dimension p. There are also techniques in terms of storage. In the BFGS method above, one can just keep track of $\mathbf{u}$, $\mathbf{v}$ and some of its (vector-valued) variations in each step, but not the entire approximated Hessian matrix. This would save a lot of memory, which is done L-BFGS (limited memory BFGS) [storing vectors save a lot of memory than saving matrices].

Example: Going back to linear regression: We now want to estimate the regression coefficient using the Newton method.

$$f(\boldsymbol{\beta}) = \frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

What are $f'(\boldsymbol{\beta})$ and $f''(\boldsymbol{\beta})$? Just modify the gradient descent code with newton update. Specifically, `derivbeta` will be replaced by $f''(\boldsymbol{\beta})^{-1}f'(\boldsymbol{\beta})$.

Debiased LASSO

We talked about debiasing LASSO estimate. Debiased lasso is actually a single-step NR update from the LASSO-derived estimate for the original objective (without the penalty).

So it's

$$\mathbf{x}^{(\text{Debiased LASSO estimate})} = \mathbf{x}^{(\text{LASSO-based estimate})} - (\bigtriangledown^2 f(\mathbf{x}^{(\text{LASSO-based estimate})}))^{-1} \bigtriangledown f(\mathbf{x}^{(\text{LASSO-based estimate})})$$

Here f is the loss-function without the LASSO penalty.