Quasi-Newton Methods

A great resource for the material in this section is [NW06, Chapter 6].

Newton's method is great in that it converges to tremendous accuracy in a very surprisingly small number of iterations, especially for smooth functions. It is not so great in that each iteration is extremely expensive. To compute the step direction,

$$\boldsymbol{d}_k = (\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k),$$

we have to

1. compute the gradient (an $N \times 1$ vector),
2. compute the Hessian (an $N \times N$ matrix),
3. invert the Hessian and apply the inverse to the gradient.

Typically, computing the gradient is reasonable (maybe $O(N^2)$ or $O(N)$ computations and storage). Computing and inverting the Hessian might be harder; in general, these operations take $O(N^3)$ computations, and this is something we will have to repeat at every iteration. If N is very large, this can be completely impractical.

At the end of the day, the quadratic model is exactly that – a model. A natural question to ask is if there are alternative quadratic models that might be cheaper while retaining the essential efficacy of Newton. There are, and they are called **quasi-Newton methods**.

Instead of calculating (and inverting) the Hessian at every point, we try to form a simple estimate of the (inverse of the) Hessian. We do this by collecting information about the curvature of the functional from the point we visit (and their gradients) as we iterate – basically, we are approximating the Hessian (the second derivative) by measuring how the gradients (the first derivative) change from point to point. What is great is that these Hessian estimates (and their inverses) can be quickly updated from one iteration to the next, thus avoiding the expensive matrix inversion.

The cost of these methods is comparable to gradient descent – along with the gradient computation, we will have to do a few matrix-vector multiplies at each iteration, the cost of which is again typically comparable to calculating $\nabla f(\boldsymbol{x}_k)$. Theoretically, their convergence properties are better than gradient descent, but not as good as Newton. In practice, they typically significantly outperform gradient descent and they are practical for problem sizes where we dare not even dream about computing the Hessian and inverting it.

Approximating the Hessian

Newton's method works by forming a quadratic model around the current iterate $\boldsymbol{x}_k$:

$$\tilde{f}_k(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \boldsymbol{g}_k \rangle + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{H}_k (\boldsymbol{x} - \boldsymbol{x}_k).$$

The particular choices of $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{H}_k = \nabla^2 f(\boldsymbol{x}_k)$ are motivated by Taylor's theorem. We minimize the surrogate functional above to compute the step direction

$$\boldsymbol{d}_k = -\boldsymbol{H}_k^{-1} \boldsymbol{g}_k,$$

choosing a step size α_k, then moving

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k.$$

We then repeat with a new quadratic model,

$$\tilde{f}_{k+1}(\boldsymbol{x}) = f(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{x} - \boldsymbol{x}_{k+1}, \boldsymbol{g}_{k+1} \rangle + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_{k+1})^{\mathrm{T}} \boldsymbol{H}_{k+1}(\boldsymbol{x} - \boldsymbol{x}_{k+1}).$$

Quasi-Newton methods operate in this same general framework, and keep the same linear term $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$. Rather than using the Hessian, we ask only that our quadratic model yield gradients that are consistent with the true gradient at both the current point $\boldsymbol{x}_{k+1}$ and the previous point $\boldsymbol{x}_k$. That is, we want

$$\nabla \tilde{f}_{k+1}(\boldsymbol{x}_{k+1}) = \nabla f(\boldsymbol{x}_{k+1}) \tag{1}$$

and

$$\nabla \tilde{f}_{k+1}(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k). \tag{2}$$

Note that

$$\nabla \tilde{f}_{k+1}(\boldsymbol{x}) = \boldsymbol{g}_{k+1} + \boldsymbol{H}_{k+1}(\boldsymbol{x} - \boldsymbol{x}_{k+1}).$$

By setting $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$, (1) will hold automatically no matter what we choose for $\boldsymbol{H}_{k+1}$. Thus, we would like to choose $\boldsymbol{H}_{k+1}$ so that (2) also holds. This will occur provided that

$$\boldsymbol{g}_{k+1} + \boldsymbol{H}_{k+1}(\boldsymbol{x}_k - \boldsymbol{x}_{k+1}) = \boldsymbol{g}_k,$$

or more compactly

$$\boldsymbol{H}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k, \tag{3}$$

where

$$\begin{aligned} \boldsymbol{s}_k &:= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \\ \boldsymbol{y}_k &:= \boldsymbol{g}_{k+1} - \boldsymbol{g}_k. \end{aligned}$$

There are many choices for $\boldsymbol{H}_{k+1}$ that satisfy (3), even if we add the constraint that it be symmetric and positive definite (which we need to ensure that $\boldsymbol{H}_{k+1}$ is invertible, allowing us to compute $\boldsymbol{d}_{k+1}$. In general, quasi-Newton methods choose $\boldsymbol{H}_{k+1}$ so that it can be easily computed from $\boldsymbol{H}_k$ – different update rules lead to different quasi-Newton methods.

BFGS

Perhaps the most widely used quasi-Newton method, and what is viewed to be the most effective, is called the BFGS[1] algorithm. BFGS is similar to many other quasi-Newton methods in that it chooses $\boldsymbol{H}_{k+1}$ to be "close" to the previous $\boldsymbol{H}_k$ in a certain sense that turns out to have computational advantages. In particular, the BFGS update can be derived as the solution to the optimization problem

$$\underset{\boldsymbol{H}}{\text{minimize}} \ \|\boldsymbol{H} - \boldsymbol{H}_k\|_{\boldsymbol{W}} \quad \text{subject to} \ \ \boldsymbol{H}^{\mathrm{T}} = \boldsymbol{H}, \quad \boldsymbol{H}\boldsymbol{s}_k = \boldsymbol{y}_k,$$

where $\|\cdot\|_{\boldsymbol{W}}$ is a particular weighted Frobenius norm (see [NW06] for details.)

It turns out that this optimization problem has a closed form solution, giving the BFGS update rule for constructing $\boldsymbol{H}_{k+1}$ from $\boldsymbol{H}_k$:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k} - \frac{\boldsymbol{H}_k\boldsymbol{s}_k(\boldsymbol{H}_k\boldsymbol{s}_k)^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k}. \tag{4}$$

At each iteration, we update $\boldsymbol{H}_k$ by adding two rank-1 matrices to $\boldsymbol{H}_k$. This is critical since in the end we need to be able to invert $\boldsymbol{H}_{k+1}$ to be able to compute the next update. In general this would

[1]Named after Broyden, Fletcher, Goldfarb, and Shanno.

remain a computational challenge, but in this case since we already know $\boldsymbol{H}_k^{-1}$ (which would have been required at the previous step) and $\boldsymbol{H}_{k+1}$ is a low-rank update to $\boldsymbol{H}_k$, there will be an efficient solution to computing $\boldsymbol{H}_{k+1}^{-1}$. As we will see below, the cost of computing $\boldsymbol{H}_{k+1}$ will be the same order as a vector-matrix multiply (i.e., $O(N^2)$ instead of $O(N^3)$).

However, before we discuss the mechanics of computing $\boldsymbol{H}_{k+1}^{-1}$, let us look a bit closer at the BFGS update in (4) and verify that it makes sense. It is easy to check that $\boldsymbol{H}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$ is always satisfied:

$$\begin{aligned}
\boldsymbol{H}_{k+1}\boldsymbol{s}_k &= \boldsymbol{H}_k\boldsymbol{s}_k + \frac{\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k} - \frac{\boldsymbol{H}_k\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k^{\mathrm{T}}\boldsymbol{s}_k}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k} \\
&= \boldsymbol{H}_k\boldsymbol{s}_k + \boldsymbol{y}_k - \boldsymbol{H}_k\boldsymbol{s}_k \\
&= \boldsymbol{y}_k,
\end{aligned}$$

where above we exploit the fact that $\boldsymbol{H}_k$ is symmetric.

It is also the case that if $\boldsymbol{H}_k$ is positive definite then $\boldsymbol{H}_{k+1}$ will also be positive definite, provided that f is *strictly* convex.[2] This follows from the monotonic gradient property of convex functions:

$$\langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\rangle > 0.$$

Setting $\boldsymbol{x} = \boldsymbol{x}_{k+1}$ and $\boldsymbol{y} = \boldsymbol{x}_k$, this tells us that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$. (This is reassuring, since if $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k = 0$ then this update rule would be somewhat problematic.)

[2]Newton and quasi-Newton algorithms are typically motivated in the context of twice differentiable functions so that the Hessian matrix always exists. Strict convexity ensures that the Hessian is always invertible, which we clearly need. If f is not strictly convex, we can actually still use the BFGS algorithm, but we need to be a bit more careful to ensure that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$ and the Hessian remains invertible. We will see later that this can instead be guaranteed as a part of the line search that selects the step size α_k.

Moreover, the fact that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$ ensures that $\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}}/\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k$ is positive semidefinite. We show below that

$$\boldsymbol{H}_k - \frac{\boldsymbol{H}_k\boldsymbol{s}_k(\boldsymbol{H}_k\boldsymbol{s}_k)^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k} \tag{5}$$

is positive semidefinite. Thus $\boldsymbol{H}_{k+1}$ is the sum of two positive semidefinite matrices, and hence positive semidefinite as well. In fact, we will be able to show that $\boldsymbol{H}_{k+1}$ is strictly positive definite by looking closely at the vectors that live in the nullspace of (5).

To see that (5) is positive semidefinite, recall that a symmetric matrix $\boldsymbol{M}$ is positive semidefinite if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$. Thus, we would like to show that

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{x} \geq \frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{x}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k}.$$

Notice that the numerator in the fraction above can be written as $(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k)^2$. A fact that you can easily verify on your own is that for any symmetric positive definite matrix $\boldsymbol{M}$, $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{y}$ defines a valid inner product. Applying the Cauchy-Schwarz inequality with this inner product yields

$$(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k)^2 \leq (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{x})(\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{H}_k\boldsymbol{s}_k)$$

and thus (5) is positive semidefinite, as desired.

From the above we have that $\boldsymbol{H}_{k+1}$ must be positive semidefinite. We can say more by looking at the eigenvalues of (5) that are zero. From the argument above it is clear that we actually have a strict inequality *unless* $\boldsymbol{x}$ is proportional to $\boldsymbol{s}_k$ (since Cauchy-Schwarz is strict unless the vectors are colinear). Said differently, the eigenvalues of (5) are all strictly positive except for one, which corresponds to the eigenvector $\boldsymbol{s}_k$. Thus, the only way that $\boldsymbol{H}_{k+1}$ could have an eigenvector of zero would be if $\boldsymbol{s}_k$ also lived in the nullspace of $\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}}/\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k$, but this is explicitly ruled out by the fact that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$. Thus, $\boldsymbol{H}_{k+1}$ must actually be positive definite.

Now, we return to the issue of calculating $\boldsymbol{H}_{k+1}^{-1}$. Noting that $\boldsymbol{H}_{k+1}$ can be expressed as the sum of $\boldsymbol{H}_k$ plus two additional terms, we can apply the Woodbury matrix identity

$$(\boldsymbol{A}+\boldsymbol{U}\boldsymbol{C}\boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1}+\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}$$

to "simplify" this inverse. After some tedious calculations we arrive at the formula:

$$\boldsymbol{H}_{k+1}^{-1} = \boldsymbol{H}_k^{-1} + \frac{(\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{y}_k + \boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{H}_k^{-1}\boldsymbol{y}_k)(\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}})}{(\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{y}_k)^2} - \frac{\boldsymbol{H}_k^{-1}\boldsymbol{y}_k\boldsymbol{s}_k^{\mathrm{T}} + \boldsymbol{s}_k\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{H}_k^{-1}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{y}_k}.$$

Note that the formula above requires computing a matrix-vector product ($\boldsymbol{H}_k^{-1}\boldsymbol{y}_k$) and computing two rank-1 matrices (scaled versions of $\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}}$ and $\boldsymbol{s}_k\boldsymbol{y}_k^{\mathrm{T}}$), but all of these computations are $O(N^2)$ as opposed to $O(N^3)$.

Above, we have spoken only about updates to the quadratic model. The BFGS algorithm requires not only an initial guess $\boldsymbol{x}_0$, but also an initial matrix $\boldsymbol{H}_0$. The most common choice here is take $\boldsymbol{H}_0 = \mathbf{I}$.

This gives us the following algorithm:

```
BFGS
  Input: $\boldsymbol{x}_0$, $\boldsymbol{H}_0^{-1}$
  Initialize: $k = 0$, $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$
  while not converged do
      $\boldsymbol{d}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$
      Select $\alpha_k$ using a line search
      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k$
      $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$
      $\boldsymbol{s} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{y} = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$, $\boldsymbol{a} = \boldsymbol{H}_k^{-1}\boldsymbol{y}$, $\gamma = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{y}$
      $\boldsymbol{H}_{k+1}^{-1} = \boldsymbol{H}_k^{-1} + \frac{\gamma + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{a}}{\gamma^2}\boldsymbol{s}\boldsymbol{s}^{\mathrm{T}} - \frac{1}{\gamma}\boldsymbol{a}\boldsymbol{s}^{\mathrm{T}} - \frac{1}{\gamma}\boldsymbol{s}\boldsymbol{a}^{\mathrm{T}}$
      $k = k + 1$
  end while
```

Convergence of BFGS

There are two main convergence results for BFGS with a step size chosen using an appropriate line search.

Global convergence: If f is strongly convex, then BFGS with backtracking converges to $\boldsymbol{x}^\star$ from any starting point $\boldsymbol{x}_0$ and initial quadratic model $\boldsymbol{H}_0 \succ \boldsymbol{0}$.

Superlinear local convergence: If f is strongly convex and the *gradient* of f is M-smooth (i.e., the Hessian is Lipschitz), then when we are close to the solution

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2 \ \leq \ c_k \|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2$$

where $c_k \to 0$.

This is not quite the quadratic convergence of the Newton method, but it can still be much, much faster than the linear rate given by gradient descent. In practice, there is often very little difference between the convergence of BFGS and Newton's method.

Line search for BFGS

We can use similar line search methods for BFGS as we have seen before in the context of gradient descent and Newton's method, with two important caveats.

First, in initializing a backtracking search it is important to set the initial step size $\bar{\alpha} = 1$. This ensures that when we get close to a solution we will be taking sufficiently large steps to ensure superlinear convergence.

Second, recall the Wolfe conditions:

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \geq c_1 \alpha \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle \tag{6}$$
$$\langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \rangle \geq c_2 \langle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k) \rangle, \tag{7}$$

where $0 < c_1 < c_2 < 1$. In the context of gradient descent, we often ignore (7) and focus only on (6) (Armijo). However, in the context of BFGS (7) also has an important role to play (especially if the objective function being minimized is not strictly convex).

Specifically, (7) guarantees that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$ at iteration k, which as discussed above ensures that the BFGS update for $\boldsymbol{H}_{k+1}$ is well-defined and guarantees that $\boldsymbol{H}_{k+1}$ remains positive definite. To see this, note that for α_k satisfying (7) we have

$$\langle \boldsymbol{d}_k, \boldsymbol{g}_{k+1} \rangle \geq c_2 \langle \boldsymbol{d}_k, \boldsymbol{g}_k \rangle,$$

which implies that

$$\langle \boldsymbol{d}_k, \boldsymbol{g}_{k+1} - \boldsymbol{g}_k \rangle \geq (c_2 - 1)\langle \boldsymbol{d}_k, \boldsymbol{g}_k \rangle.$$

Note $c_2 < 1$, so that $(c_2 - 1) < 0$. Moreover, since $\boldsymbol{d}_k$ is a descent direction, we have that $\langle \boldsymbol{d}_k, \boldsymbol{g}_k \rangle < 0$, and thus the right-hand side above is strictly positive. Thus

$$\langle \boldsymbol{d}_k, \boldsymbol{g}_{k+1} - \boldsymbol{g}_k \rangle = \langle \boldsymbol{d}_k, \boldsymbol{y}_k \rangle > 0.$$

Since $\boldsymbol{s}_k = \alpha_k \boldsymbol{d}_k$, this also shows that $\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k > 0$, as desired.

References

[NW06] J. Nocedal and S. Wright. *Numerical Optimization.* Springer, 2nd edition, 2006.