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,

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

we have to

1. compute the gradient (an N 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 $O(N)$ flops and storage). Computing and inverting the Hessian might be harder; in general, these operations take $O(N^3)$ flops ... and that is for every iteration.

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 estimate 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 (extremely) 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 significantly outperform gradient descent and they are practical for problem sizes where we dare not even dream about computing the Hessian and inverting it.

We will look at one particular quasi-Newton method, the BFGS algorithm[1]. It is based on one of my favorite tricks in linear algebra: the low-rank update.

Low rank updates

Suppose that I have computed the inverse $\boldsymbol{P}^{-1}$ of an $N \times N$ symmetric matrix $\boldsymbol{P}$. If I "update" $\boldsymbol{P}$ by adding a low rank symmetric matrix $\boldsymbol{L}$, then inverse $(\boldsymbol{P}+\boldsymbol{L})^{-1}$ can be computed in $O(RN^2)$ time (where R is the rank of $\boldsymbol{L}$). This is faster than the $O(N^3)$ time to invert the matrix from scratch, especially when $R \ll N$.

For example, suppose I have $\boldsymbol{P}^{-1}$ on hand, and I would like to compute $(\boldsymbol{P}+\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}})^{-1}$ for some given matrix $\boldsymbol{v}$. Simply writing this out, you will see that this boils down to a comparable rank-1 update of

[1]The co-inventors are Broyden, Fletcher, Goldfarb, and Shanno.

the inverse:

$$(\boldsymbol{P} + \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}})^{-1} = \boldsymbol{P}^{-1} - \frac{1}{1 + \boldsymbol{v}^{\mathrm{T}}\tilde{\boldsymbol{v}}}\tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^{\mathrm{T}}, \quad \text{where } \ \tilde{\boldsymbol{v}} = \boldsymbol{P}^{-1}\boldsymbol{v}.$$

This means that given $\boldsymbol{P}^{-1}$, the cost of computing $(\boldsymbol{P} + \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}})^{-1}$ is on the same order as a vector-matrix multiply (i.e. $O(N^2)$ instead of $O(N^3)$).

In general, if $\boldsymbol{U}$ and $\boldsymbol{V}$ are $N \times R$ matrices, then

$$(\boldsymbol{P} + \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}})^{-1} = \boldsymbol{P}^{-1} - \tilde{\boldsymbol{U}}(\mathbf{I} + \boldsymbol{V}^{\mathrm{T}}\tilde{\boldsymbol{U}})^{-1}\tilde{\boldsymbol{V}}^{\mathrm{T}},$$

where

$$\tilde{\boldsymbol{U}} = \boldsymbol{P}^{-1}\boldsymbol{U}, \quad \tilde{\boldsymbol{V}} = \boldsymbol{P}^{-1}\boldsymbol{V}.$$

This is a specialized version of the *Sherman-Morrison-Woodbury* identity. Note that computing $\tilde{\boldsymbol{U}}$ and $\tilde{\boldsymbol{V}}$ costs $O(RN^2)$, while computing the matrix inverse costs $O(R^3)$ (which is actually cheaper because $R \leq N$).

Approximate Hessian updates: DFP and BFGS

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

$$\tilde{f}_k(\boldsymbol{x}^{(k)} + \boldsymbol{v}) = f(\boldsymbol{x}^{(k)}) + \langle \boldsymbol{v}, \boldsymbol{a}_k \rangle + \frac{1}{2}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{P}_k\boldsymbol{v}.$$

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

$$\boldsymbol{d}^{(k+1)} = -\boldsymbol{P}_k^{-1}\boldsymbol{a}_k,$$

choosing a step size t_{k+1}, then moving

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + t_{k+1}\boldsymbol{d}^{(k+1)}.$$

We then repeat with a new quadratic model,

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

Quasi-newton methods operate in this same general framework, and keep the same linear term $\boldsymbol{a}_k = \nabla f(\boldsymbol{x}^{(k)})$. Rather than compute the Hessian, we ask only that our quadratic model yield gradients that are consistent with gradient we have computed at the current point, and the gradient we saw at the previous iteration. That is, we want

$$\nabla \tilde{f}_{k+1}(\boldsymbol{x}^{(k+1)}) = \nabla f(\boldsymbol{x}^{(k+1)}), \quad \text{and} \quad \nabla \tilde{f}_{k+1}(\boldsymbol{x}^{(k)}) = \nabla f(\boldsymbol{x}^{(k)}).$$

Using the gradients for the $\boldsymbol{a}_k$ in the linear terms, the first condition above is automatic no matter what we choose for $\boldsymbol{P}_{k+1}$, as we can see by taking $\boldsymbol{v} = \boldsymbol{0}$ above. So we would like to choose $\boldsymbol{P}_{k+1}$ so that the second condition above holds. This means we need that

$$\nabla \tilde{f}_{k+1}(\boldsymbol{x}^{(k+1)} - t_{k+1}\boldsymbol{d}^{(k+1)}) = \nabla f(\boldsymbol{x}^{(k)}),$$

meaning we need to choose $\boldsymbol{P}_{k+1}$ so that,

$$t_{k+1}\boldsymbol{P}_{k+1}\boldsymbol{d}^{(k+1)} = \nabla f(\boldsymbol{x}^{(k+1)}) - \nabla f(\boldsymbol{x}^{(k)}).$$

Since $t_{k+1}\boldsymbol{d}^{(k+1)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$, we can write this condition compactly as

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

where

$$\boldsymbol{s}_k = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}, \quad \boldsymbol{y}_k = \nabla f(\boldsymbol{x}^{(k+1)}) - \nabla f(\boldsymbol{x}^{(k)}).$$

.

There are, of course, many choices for $\boldsymbol{P}_{k+1}$ that satisfy (**??**), even if we add the constraint that it be symmetric positive semidefinite. In general, quasi-Newton methods choose the $\boldsymbol{P}_{k+1}$ that is "closest" to the last quadratic model $\boldsymbol{P}_k$ by solving

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

where $\|\cdot\|_M$ is some matrix norm — different norms lead to different quasi-Newton methods.

DFP

The original quasi-Newton method, developed by Davidson in the 50s, then analyzed by Fletcher and Powell, is based on a using a certain weighted Frobenius norm for $\|\cdot\|_M$ above. The update, for which you can check satisfies (1), is

$$\boldsymbol{P}_{k+1}=(\mathbf{I}-\gamma_k\boldsymbol{y}_k\boldsymbol{s}_k^{\mathrm{T}})\boldsymbol{P}_k(\mathbf{I}-\gamma_k\boldsymbol{s}_k\boldsymbol{y}_k^{\mathrm{T}})+\gamma_k\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}} \quad \text{(DFP)},$$

where $\gamma_k=\frac{1}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k}$. This is called the DFP update, and we move from $\boldsymbol{P}_k$ to $\boldsymbol{P}_{k+1}$ by adding a rank-2 matrix. This step corresponds to finding the matrix that is closest to $\boldsymbol{P}_k$ in a certain norm under the constraint $\boldsymbol{P}_{k+1}\boldsymbol{s}_k=\boldsymbol{y}_k$. But basically, you are performing an update to remove the parts of the row and column spaces of $\boldsymbol{P}_k$ that are $\boldsymbol{s}_k$ (note that $\boldsymbol{s}_k$ is in the null space of the first matrix above), then replacing that with a carefully chosen rank-1 matrix so that the constraint is met.

Letting $\boldsymbol{Q}_k=\boldsymbol{P}_k^{-1}$, we can apply the Woodbury formula above to yield

$$\boldsymbol{Q}_{k+1} = \boldsymbol{Q}_k - \frac{1}{\boldsymbol{y}_k^{\mathrm{T}}\tilde{\boldsymbol{y}}_k}\tilde{\boldsymbol{y}}_k\tilde{\boldsymbol{y}}_k^{\mathrm{T}} + \frac{1}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k}\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}}, \qquad \text{(DFP)}$$

where $\tilde{\boldsymbol{y}}_k = \boldsymbol{Q}_k\boldsymbol{y}_k$.

BFGS

Perhaps the most widely used quasi-Newton methods, and what is viewed to be the most effective, is called the BFGS[2] algorithm. It is very similar to the DFP updated above; the only difference is that the inverse $\boldsymbol{Q}_{k+1}$ is updated directly from $\boldsymbol{Q}_k$ while maintaining the constraint $\boldsymbol{Q}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$. The update below can be interpreted as choosing among all *inverse* matrices that are closest to $\boldsymbol{P}_k^{-1}$ such that (1) is satisfied. The update is

$$\boldsymbol{Q}_{k+1} = (\mathbf{I} - \gamma_k\boldsymbol{s}_k\boldsymbol{y}_k^{\mathrm{T}})\boldsymbol{Q}_k(\mathbf{I} - \gamma_k\boldsymbol{y}_k\boldsymbol{s}_k^{\mathrm{T}}) + \gamma_k\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}} \quad \text{(BFGS)},$$

where again $\gamma_k = \frac{1}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k}$. For computing the step direction, the update above is all we need (since we will take $\boldsymbol{d}^{(k+1)} \propto -\boldsymbol{Q}_{k+1}\nabla f(\boldsymbol{x}^{(k)})$). But for good measure, and to compare it more directly to the DFP update, we can again apply Woodbury to see that this corresponds to the following update to $\boldsymbol{P}_{k+1}$ in the quadratic model:

$$\boldsymbol{P}_{k+1} = \boldsymbol{P}_k - \frac{1}{\boldsymbol{s}_k^{\mathrm{T}}\tilde{\boldsymbol{s}}_k}\tilde{\boldsymbol{s}}_k\tilde{\boldsymbol{s}}_k^{\mathrm{T}} + \frac{1}{\boldsymbol{y}_k^{\mathrm{T}}\boldsymbol{s}_k}\boldsymbol{y}_k\boldsymbol{y}_k^{\mathrm{T}}, \qquad \text{(BFGS)}$$

with $\tilde{\boldsymbol{s}}_k = \boldsymbol{P}_k\boldsymbol{s}_k$.

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 sym+def matrix $\boldsymbol{P}_0$. It is usually sufficient to take $\boldsymbol{P}_0 = \mathbf{I}$.

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

It is a fact that the BFGS update maintains the positive-semidef initeness of the $\boldsymbol{P}_k$ and $\boldsymbol{Q}_k$. It is worth sitting down and working out why this is true.

Convergence of BFGS

There are two main convergence results for BFGS with a step size chosen using a backtracking line search — for proof of these, see [NW06].

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 $\boldsymbol{Q}_0 \succ \boldsymbol{0}$.

Superlinear local convergence: If f is strongly convex and $\nabla^2 f(\boldsymbol{x})$ 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 is certainly faster than the linear rate given by gradient descent. In practice, there is often times very little difference between the accuracy of BFGS and the Newton method.

References

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