Accelerated first-order methods

There are small changes we can make to gradient descent that can dramatically improve its performance, both in theory and in practice. We talk about two of these here: the heavy ball method, and Nesterov's "optimal algorithm".

The heavy ball method

The **heavy ball method**, introduced by Polyak in 1964, introduces an additional term term into the gradient step:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha_k \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) + \beta_k (\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}).$$

The second term above adds a little bit of the last step $\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}$ direction into the new step direction $\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$. We can think of this as a mixture of updating our step direction but also biasing in the direction that we were already going, much like a (heavy) ball would as it rolls down a surface and builds momentum. This method is also referred to as *gradient descent with momentum*.

In the last set of notes we discussed the convergence of gradient descent when f is both M-smooth and strongly convex. A key result was that for a fixed step size of $\alpha = 1/M$, we have

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2,$$

where $\kappa = M/m$.

For the heavy ball method, we have a similar analysis that ends in a better result. Specifically, one can show that taking

$$\alpha = \frac{4}{(\sqrt{M} - \sqrt{m})^2}, \quad \beta = \frac{\sqrt{M} - \sqrt{m}}{\sqrt{M} + \sqrt{m}},$$

yields a guarantee of the form

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2 \lesssim \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2.$$

The notation "$\lesssim$" indicates that we are ignoring a (small) constant to highlight the dependence on k and κ. This can be translated into a guarantee that says

$$\frac{\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2}{\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2} \leq \epsilon \quad \text{when} \quad k \gtrsim \sqrt{\kappa}\,\log(1/\epsilon).$$

The difference with gradient descent can be significant. When $\kappa = 10^2$, we are asking for $\approx 100\log(1/\epsilon)$ iterations for gradient descent, as compared with $\approx 10\log(1/\epsilon)$ from the heavy ball method.

Nesterov's "optimal" method

In the case where f is strictly convex, there are examples that show that the convergence rate of the heavy ball method can't be improved in general. For non-strictly convex f, the story is more complicated.

Recall that we also had a convergence result for gradient descent in the case where we only have that f is M-smooth. Specifically, again setting $\alpha = 1/M$, we showed that

$$f(\boldsymbol{x}^{(k)}) - f(\boldsymbol{x}^\star) \lesssim \frac{M}{k}\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2^2.$$

Thus, to reduce the error by a factor of ϵ requires

$$k \gtrsim 1/\epsilon$$

iterations.

In 1983, Yuri Nesterov proposed a slight variation on the heavy ball method that can improve on this theory, and often works better in practice.[1] Specifically, note that the heavy ball method can be represented via the iteration:

$$\begin{aligned}\boldsymbol{p}^{(k)} &= \beta_k \left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} - \alpha_k \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) + \boldsymbol{p}^{(k)},\end{aligned}$$

where we start with $\boldsymbol{p}^{(0)} = \mathbf{0}$. Nesterov's method makes a subtle, but significant, change to this iteration:

$$\begin{aligned}\boldsymbol{p}^{(k)} &= \beta_k \left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} - \alpha_k \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)} + \boldsymbol{p}^{(k)}) + \boldsymbol{p}^{(k)}.\end{aligned}$$

Notice that this is the same as heavy ball *except* that there is also a momentum term *inside* the gradient expression. With this iteration, using a very similar analysis to what we did with gradient descent, you can show that by carefully selecting β_k we can guarantee that

$$f(\boldsymbol{x}^{(k)}) - f(\boldsymbol{x}^\star) \lesssim \frac{L}{k^2} \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2^2.$$

This means that we can reduce the error by a factor of ϵ in

$$k \gtrsim 1/\sqrt{\epsilon},$$

iterations. When $\epsilon \sim 10^{-4}$, this is much, much better than the $1/\epsilon$ we had for gradient descent.

[1] Note that this method remained to a large extent unknown in the wider community until the 2004 publication (in English) of his book *Introductory Lectures on Convex Optimization*.

Nesterov's method is called "optimal" because it is impossible to beat the $1/k^2$ rate using only function and gradient evaluations. There are careful demonstrations of this in the literature.

Note that in practice, α_k can be chosen using a standard line search, and a good choice of β_k (both in theory and in practice) turns out to be

$$\beta_k = \frac{k-1}{k+2}.$$

Most significantly, note that in setting β_k we do *not* need to know anything about the function we are minimizing (such as strong convexity parameters). This represents an important advantage compared to the heavy ball method described before.

Newton's Method

So far we have focused on what are called *first order methods*, that is, optimization methods that assume our function is differentiable and work by iterative taking the (first) derivative of the function to calculate the appropriate step direction. For functions that are twice-differentiable, we can sometimes dramatically reduce the number of iterations required by exploiting *second order* information (i.e., the second derivative).

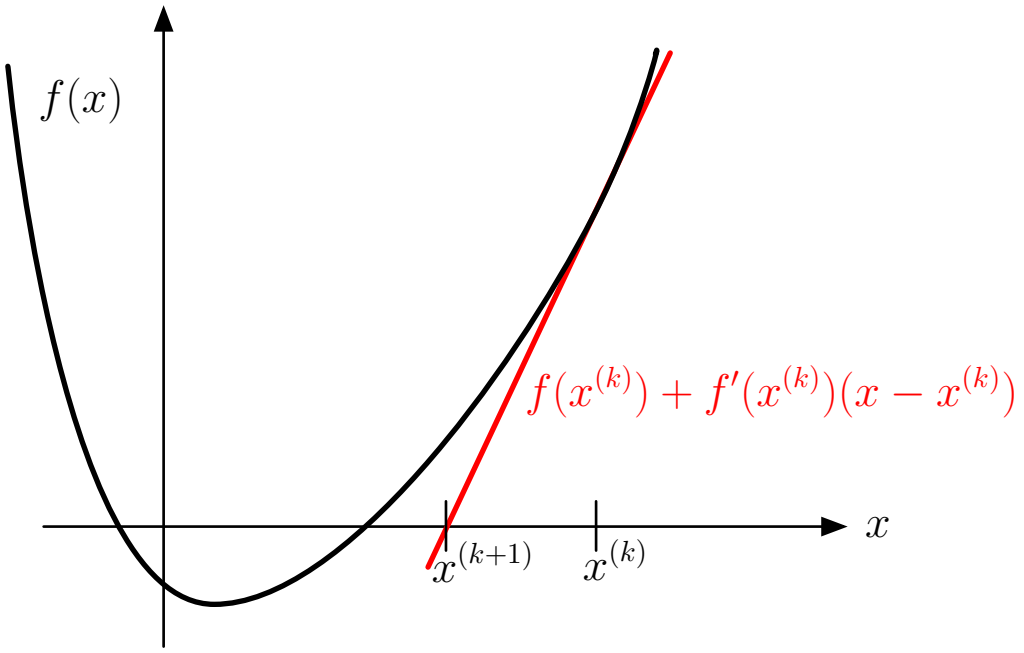
The most common second order approach is based on **Newton's method**, which is a classical technique for finding the root of a general differentiable function $f(x) : \mathbb{R} \to \mathbb{R}$. Specifically, suppose that we want to find an $x \in \mathbb{R}$ such that

$$f(x) = 0.$$

You may recall that one technique for doing this is to start at some guess x_0, and then follow the iteration

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}.$$

This iteration comes from taking a linear approximation at $x^{(k)}$ and then finding the root for this approximation, as illustrated below:



Of course, there can be many roots, and which one we converge to will depend on what we choose for x_0. It is also very much possible that the iterations do not converge for some initial values x_0. But there is a classical result that says that once we are "close enough" to a particular root x_0, we will have

$$\underbrace{|x_0 - x^{(k+1)}|}_{\epsilon_{k+1}} \leq C \cdot \underbrace{(x_0 - x^{(k)})^2}_{\epsilon_k^2},$$

where the constant C depends on the ratio between the first and second derivatives in an interval around x_0.[2] The take-away here is that close to the solution, Newton's methods exhibits *quadratic convergence*: the error at the next iteration is proportional to the *square* of the error at the last iteration. Since we are concerned with ϵ_k small, $\epsilon_k \ll 1$, this means that under the right conditions, the error goes down in dramatic fashion from iteration to iteration.

When $f(x)$ is convex, twice differentiable, and has a minimizer, we can find a minimizer by applying Newton's method to the derivative, since finding a root of the derivative is the same as finding a minimizer. We start at some initial guess $x^{(0)}$, and then take

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}. \tag{1}$$

We can interpret the iteration (1) above in the following way:

1. At $x^{(k)}$, approximate $f(x)$ using the Taylor expansion
$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}f''(x^{(k)})(x - x^{(k)})^2.$$
2. Find the exact minimizer of this quadratic approximation. Taking the derivative of the expansion above and setting it equal to zero yields the following optimality condition for $x^{(k+1)}$ to be a minimizer:
$$(x^{(k+1)} - x^{(k)})f''(x^{(k)}) = -f'(x^{(k)}).$$
This is just a re-arrangement of the iteration (1).

[2] There are various technical conditions that f must obey for this result to hold, including the second derivative being continuous and the first derivative not being equal to zero. Also, the condition "close enough" is characterized by looking at ratios of derivatives on an interval around x_0.

This last interpretation extends naturally to the case where $f(\boldsymbol{x})$ is a function of many variables, $f : \mathbb{R}^N \to \mathbb{R}$. We know that if f is convex and twice differentiable, we have a minimizer $\boldsymbol{x}^\star$ when $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^\star) = \boldsymbol{0}$. Newton's method to find such a minimizer proceeds as above. We start with an initial guess $\boldsymbol{x}^{(0)}$, and use the following iteration:

1. Take a Taylor approximation around $f(\boldsymbol{x}^{(k)})$:

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

   where $\boldsymbol{g} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)})$ and $\boldsymbol{H} = \boldsymbol{D}_f^2(\boldsymbol{x}^{(k)})$.

2. Find the exact minimizer to this approximation:

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^{(k)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(k)})^{\mathrm{T}} \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}^{(k)}).$$

   Since $\boldsymbol{H}$ is symmetric and positive semi-definite, we know that the conditions for $\boldsymbol{x}^{(k+1)}$ being a minimizer[3] are

$$\boldsymbol{H}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) = -\boldsymbol{g}.$$

   If $\boldsymbol{H}$ is invertible (i.e., it is strictly positive definite), then we have a unique minimizer and set

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{H}^{-1}\boldsymbol{g}.$$

To summarize, this gives us an update of the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left(\boldsymbol{D}_f^2(\boldsymbol{x}^{(k)})\right)^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}).$$

[3] Take the gradient of this new expression and set it equal to $\boldsymbol{0}$.

This procedure is often referred to as a *pure Newton step*, as it does not involve the selection of a step size. In practice, however, it is often beneficial to choose the step direction as

$$\boldsymbol{d}^{(k)} = -\left(\boldsymbol{D}_f^2(\boldsymbol{x}^{(k)})\right)^{-1} \nabla f(\boldsymbol{x}^{(k)}),$$

and then choose a step size α_k using a backtracking line search, and then take

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

as before.

Quasi-Newton Methods

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 can be extremely expensive. To compute the step direction,

$$\boldsymbol{d}^{(k)} = \left(\boldsymbol{D}_f^2(\boldsymbol{x}^{(k)})\right)^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}),$$

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 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 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 $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)})$. 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.

Approximating the Hessian

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

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

The particular choices of $\boldsymbol{g}_k = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)})$ and $\boldsymbol{H}_k = \boldsymbol{D}_f^2(\boldsymbol{x}^{(k)})$ are motivated by Taylor's theorem. We minimize the surrogate func-

tional 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,

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

Quasi-newton methods operate in this same general framework, and keep the same linear term $\boldsymbol{g}_k = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)})$. 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

$$\nabla_{\boldsymbol{x}} \tilde{f}_{k+1}(\boldsymbol{x}^{(k+1)}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k+1)})$$

and

$$\nabla_{\boldsymbol{x}} \tilde{f}_{k+1}(\boldsymbol{x}^{(k)}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}).$$

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

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

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

$$\alpha_k \boldsymbol{H}_{k+1}\boldsymbol{d}^{(k)} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k+1)}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}).$$

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

$$\boldsymbol{H}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k, \tag{2}$$

where

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

There are many choices for $\boldsymbol{H}_{k+1}$ that satisfy (2), 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 methods, and what is viewed to be the most effective, is called the BFGS[4] algorithm. BFGS uses the following update rule for constructing $\boldsymbol{H}_{k+1}$ from $\boldsymbol{H}_k$:

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

At each iteration, we update $\boldsymbol{H}_k$ by adding two rank-1 matrices to $\boldsymbol{H}_k$. These are carefully chosen to ensure that the constraint $\boldsymbol{H}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$ is always satisfied. This is easy to check:

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

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

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.[5] This follows from an elementary fact about convex functions that we have not used in this class so far. Specifically, recall that if f is differentiable and strictly convex, then we have that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^N$

$$f(\boldsymbol{x}) > f(\boldsymbol{x}') + \langle \boldsymbol{x} - \boldsymbol{x}', \nabla_{\boldsymbol{x}} f(\boldsymbol{x}') \rangle,$$

and similarly we must also have that

$$f(\boldsymbol{x}') > f(\boldsymbol{x}) + \langle \boldsymbol{x}' - \boldsymbol{x}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \rangle.$$

Adding these two together and rearranging gives

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

This holds for any $\boldsymbol{x}, \boldsymbol{x}'$, but note that if we set $\boldsymbol{x} = \boldsymbol{x}^{(k+1)}$ and $\boldsymbol{x}' = \boldsymbol{x}^{(k)}$, then 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.)

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 definite. We can also show that

$$\boldsymbol{H}_k - \frac{\boldsymbol{H}_k \boldsymbol{s}_k (\boldsymbol{H}_k \boldsymbol{s}_k)^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{H}_k \boldsymbol{s}_k} \tag{3}$$

[5] 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. Here we use strict convexity to show that $\boldsymbol{y}_k^{\mathrm{T}} \boldsymbol{s}_k > 0$. In the case of functions that are not strictly convex, this condition can be ensured instead as a part of the line search that selects the step size α_k.

is positive semidefinite, and thus $\boldsymbol{H}_{k+1}$ must be positive definite.

To see that (3) 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 $\boldsymbol{x} \neq \mathbf{0}$. 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 (3) is positive semidefinite, as desired.

In practice, what we actually need at each iteration is $\boldsymbol{H}_k^{-1}$. It turns out that if $\boldsymbol{H}_{k+1}$ is a low-rank perturbation of $\boldsymbol{H}_k$, there is also a simple and efficient way to compute $\boldsymbol{H}_{k+1}^{-1}$ from $\boldsymbol{H}_k^{-1}$. In particular, a tedious calculation using the Woodbury matrix identity[6] gives the formula:

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

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 $\boldsymbol{H}_0 = \mathbf{I}$.

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

This gives us the following algorithm:

BFGS

Input: $\boldsymbol{x}^{(0)}$, $\boldsymbol{H}_0^{-1}$.

Initialize: $k = 0$, $\boldsymbol{g}_0 = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)})$

while not converged **do**

$\boldsymbol{d}^{(k)} = -\boldsymbol{H}_k^{-1} \boldsymbol{g}_k$

Select α_k using a line search

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

$\boldsymbol{g}_{k+1} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k+1)})$

$\boldsymbol{s} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$, $\boldsymbol{y} = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$, $\boldsymbol{a} = \boldsymbol{H}_k^{-1} \boldsymbol{y}$, $\gamma = \boldsymbol{s}^{\mathrm{T}} \boldsymbol{y}$

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

$k = k + 1$

end while

Convergence of BFGS

There are two main convergence results for BFGS with a step size chosen using a backtracking 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 Lipshitz), 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 times very little difference between the convergence of BFGS and Newtons method.