1 Motivation

Previously, we assumed that computing the gradient $\nabla f(x)$ or the Hessian matrix $Hf(x)$ was easy and costless. But in practice, there are many situations where that’s not true:

- If $f : \mathbb{R}^n \to \mathbb{R}$ with $n$ large, then computing $Hf(x)$ requires finding many derivatives, which is expensive even when we can take partial derivatives of $f$ easily.

- If $f$ is not given to us explicitly, then we cannot compute derivatives of $f$ directly (unless we use some approximation algorithm for derivatives).

Broyden’s method is an algorithm meant to replace Newton’s method (for solving an equation, not for minimizing) under these circumstances.

To get some intuition for this, let’s first talk about the secant method: a method that works in one dimension. Here, we want to replace the iterative step of Newton’s method,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

by some step that does not involve computing $f'(x_k)$.

Since we have already computed $f(x_k)$ and $f(x_{k-1})$, it makes sense to use these to approximate the derivative: instead of $f'(x_k)$, we can use $\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$. This gives us the iterative step

$$x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}(x_k - x_{k-1}),$$

which involves no derivatives. That’s the secant method.

As we approach a solution $x^*$, $x_k$ and $x_{k-1}$ become very close to each other, and therefore $\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$ is a very good approximation of $f'(x_k)$ (or of $f'(x_{k-1})$). So we expect this method to perform almost as well as Newton’s method.

Of course, this means that we’ll run into the same problems that we ran into with Newton’s method: slow convergence to $x^*$ when $f'(x^*) = 0$, no guarantees on convergence when $f'(x^*)$ does not exist, no guarantees on convergence far away from $x^*$, and awkward situations at some isolated points. (In the case of the secant method, we’ll have trouble in the rare case that $f(x_k) = f(x_{k-1})$, since then we’re dividing by 0.)

---

1This document comes from the Math 484 course webpage: https://faculty.math.illinois.edu/~mlavrov/courses/484-fall-2018.html
2 Rank one updates

Now let’s go to the $n$-dimensional case: for a function $g: \mathbb{R}^n \to \mathbb{R}^n$, we want to find a point $x^* \in \mathbb{R}^n$ satisfying $g(x^*) = 0$.

The first problem we run into when we try to generalize the secant method is that knowing $g(x^{(k)})$ and $g(x^{(k-1)})$ does not give us enough information to try to estimate the Jacobian $\nabla g(x^{(k)})$. Intuitively, we can learn something about the rate of change of $g$ in the direction of $x^{(k)} - x^{(k-1)}$, but nothing about the rate of change of $g$ in other directions.

Our solution to this is to keep around a matrix that approximates the Jacobian, and update it at every step with the new information we learn, instead of throwing it away entirely as the secant method does.

Let’s build on the justification behind Newton’s method to figure out how to do this. With Newton’s method, when we’re at a point $x^{(k)}$, we make a linear approximation to $g$:

$$g(x) \approx g(x^{(k)}) + \nabla g(x^{(k)})(x - x^{(k)}).$$

Then we let $x^{(k+1)}$ be the point where the linear approximation equals 0.

Now suppose that instead of computing the Jacobian $\nabla g(x^{(k)})$, we somehow found an approximation $D_k$: an $n \times n$ matrix that’s our best guess at the partial derivatives of $g$. Just as with Newton’s method, we make a linear approximation

$$g(x) \approx g(x^{(k)}) + D_k(x - x^{(k)}).$$

We let $x^{(k+1)}$ be the point where the linear approximation equals 0.

In practice, $g(x^{(k+1)})$ doesn’t end up being 0: if everything is going well, it should be closer to 0 than $g(x^{(k)})$ was, but the linear approximation is not exact. This is new information, and we should replace $D_k$ by a new matrix $D_{k+1}$ that takes this new information into account.

It’s natural to ask that $D_{k+1}$ predict correctly what $D_k$ was wrong about. We now know the value of $g(x^{(k+1)})$, so we can ask that if we replace $D_k$ by $D_{k+1}$ in our previous approximation, it should give that value exactly. That is, we should have

$$g(x^{(k)}) + D_{k+1}(x^{(k+1)} - x^{(k)}) = g(x^{(k+1)}) \quad (1)$$

where we previously had

$$g(x^{(k)}) + D_k(x^{(k+1)} - x^{(k)}) = 0. \quad (2)$$

Second, we ask that in every direction orthogonal to the direction $x^{(k+1)} - x^{(k)}$, we still make the same prediction. After all, we didn’t go in such directions, so we can’t have learned anything new about them. So we require that

$$D_k y = D_{k+1} y \text{ whenever } y \cdot (x^{(k+1)} - x^{(k)}) = 0. \quad (3)$$

Together, equations (1), (2), and (3) characterize the change from $D_k$ to $D_{k+1}$.
It is easier to reason in terms of the “update” from $D_k$ to $D_{k+1}$: the difference between $D_{k+1}$ and $D_k$. If we denote this difference by $U_k = D_{k+1} - D_k$, and define $b^{(k)} = x^{(k+1)} - x^{(k)}$, then by taking the difference of (1) and (2), we get

$$U_k b^{(k)} = g(x^{(k+1)})$$

while (3) can be rewritten as

$$U_k y = 0 \text{ whenever } y \cdot b^{(k)} = 0.$$

If this matrix exists, it must be unique, because any input $y$ can be written as $y = y^\parallel + y^\perp$ where $y^\parallel$ is a multiple of $b^{(k)}$, and $y^\perp$ is orthogonal to $b^{(k)}$; the equations above define what $U_k$ does to $y^\parallel$ and $y^\perp$, and therefore they determine what $U_k$ does to $y$.

One way to get a function $f(y)$ that has this property is to scale $g(x^{(k+1)})$ (the desired nonzero output) proportionally to the dot product $y \cdot b^{(k)}$. That is, to set

$$f(y) = \frac{y \cdot b^{(k)}}{b^{(k)} \cdot b^{(k)}} g(x^{(k+1)}).$$

This turns out to be a linear function, so that there actually is some matrix $U_k$ such that $f(y) = U_k y$. We can see this by rewritesing $y \cdot b^{(k)}$ as $(b^{(k)})^T y$, so that

$$f(y) = \frac{g(x^{(k+1)})(b^{(k)})^T y}{b^{(k)} \cdot b^{(k)}}.$$

This tells us that the unique update matrix $U_k$ that does the job is

$$U_k = \frac{g(x^{(k+1)})(b^{(k)})^T}{b^{(k)} \cdot b^{(k)}}.$$

Adding this matrix to $D_k$ to get $D_{k+1}$ is called a rank-one update, because the rank of the matrix $U_k$ is 1: its columns are all multiples of $g(x^{(k+1)})$. (And its rows are all multiples of $(b^{(k)})^T$.) This makes it a “minimal” change from $D_k$ to $D_{k+1}$ in some sense, and using a rank-one matrix also has some theoretical benefits.

### 3 Broyden’s method

#### 3.1 The method

We begin, as usual, with some starting point $x^{(0)}$; we also need a matrix $D_0$ to start as our approximation to $\nabla g(x^{(0)})$. (If computing derivatives of $g$ is expensive but not impossible, we could set $D_0$ equal to the Jacobian, since we only need to do this once. Otherwise, we could set $D_0$ to the identity matrix because that’s simple to work with.)

To compute $x^{(k+1)}$ and $D_{k+1}$ from $x^{(k)}$ and $D_k$, we:

1. Solve $g(x^{(k)}) + D_k(x^{(k+1)} - x^{(k)}) = 0$ for $x^{(k+1)}$; equivalently, set $x^{(k+1)} = x^{(k)} - D_k^{-1} g(x^{(k)})$.

2. Set $D_{k+1} = D_k + \frac{g(x^{(k+1)})(b^{(k)})^T}{b^{(k)} \cdot b^{(k)}}$, where $b^{(k)} = x^{(k+1)} - x^{(k)}$. 

3. 

3.2 An example

Here’s an example that illustrates how the method works, and also demonstrates the ability of Broyden’s method to recover from a bad initial guess for $D_0$.

Suppose we are solving the system of equations

\[
\begin{align*}
x + y &= 2, \\
x - y &= 0
\end{align*}
\]

and decide that taking the derivatives of these linear functions is too hard for us. So we’re going to pick an initial guess $(x_0, y_0) = (0, 0)$ and and set $D_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Our function $g$ is $g(x, y) = (x + y - 2, x - y)$.

Our first step sets

\[ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - D_0^{-1} g(x_0, y_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \]

so the step we took is $b^{(0)} = (2, 0)$, and $g(x_1, y_1) = (0, 2)$. Using the update formula

\[ D_1 = D_0 + \frac{g(x_1, y_1)(b^{(0)})^T}{b^{(0)} \cdot b^{(0)}} \]

we set

\[ D_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 0 \\ 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \]

(Newton’s method, or even Broyden’s method with an accurate $D_0$, would have gotten the answer in this first step, but we’ll need a bit more work.)

Our second step sets

\[ \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \]

so the step we took is $b^{(1)} = (0, -2)$ and $g(x_2, y_2) = (-2, 4)$. We do another rank-one update to get

\[ D_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} -2 \\ 0 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \]

Now, with information at more points, $D_2$ becomes the correct Jacobian matrix of the linear function $g$. In the next step, our linear approximation to $g$ will actually be $g$, and so $(x_3, y_3)$ will be the correct answer $(1, 1)$.

In general, it is not always true that after a bad guess of $D_0$, the matrix $D_k$ will always approximate $\nabla g(x^{(k)})$ after many steps. It is often the case that after enough steps, the matrix $D_k$ will accurately tell us what $\nabla g(x^{(k)})$ does in the relevant directions: the ones we actually need to use.
4 The Sherman–Morrison formula

The material in this section is not covered in the textbook, but it’s a significant factor in why Broyden’s method is computationally efficient, and explains why we’re so excited that the update matrix \( U_k \) is a rank-one matrix.

When we were using Newton’s method, we wanted to avoid computing \( \nabla g(x^{(k)})^{-1} \); instead, we preferred to solve a system of linear equations at each step. With Broyden’s method, however, the iterative step

\[
 x^{(k+1)} = x^{(k)} - D_k^{-1} g(x^{(k)})
\]

is actually much faster to use. This is because we don’t have to recompute the inverse from scratch after a rank-one update: we can find \( D_{k+1}^{-1} \) directly from \( D_k^{-1} \), by a result known as the Sherman–Morrison formula.

The Sherman–Morrison formula says that if \( A \) is an \( n \times n \) invertible matrix and \( u, v \in \mathbb{R}^n \), then \((A + uv^T)^{-1}\) can be computed from \( A^{-1} \) as

\[
 (A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}.
\]

We can use this to compute \( D_{k+1}^{-1} \) from \( D_k^{-1} \) if we set \( A = D_k \), \( u = g(x^{(k+1)}) \), and \( v = \frac{b^{(k)}}{g(x^{(k+1)})} \).

Since the Sherman–Morrison formula never requires multiplying two matrices together (we can write \( A^{-1}uv^TA^{-1} \) as \( A^{-1}u \times v^T A^{-1} \)), applying it is much faster even than solving a system of linear equations.

In this way, we avoid ever having to compute \( D_k \) explicitly: only \( D_k^{-1} \) is needed. After simplifying the Sherman–Morrison formula to our specific needs, we can re-summarize Broyden’s method as:

1. Set \( x^{(k+1)} = x^{(k)} - D_k^{-1} g(x^{(k)}) \).
2. In terms of the vectors \( b^{(k)} = x^{(k+1)} - x^{(k)} = -D_k^{-1} g(x^{(k)}) \) and \( c^{(k)} = D_k^{-1} g(x^{(k+1)}) \), set

\[
 D_{k+1}^{-1} = D_k^{-1} - \frac{c^{(k)} (b^{(k)})^T D_k^{-1}}{b^{(k)} \cdot (b^{(k)} + c^{(k)})}.
\]

Even if the evaluation of \( \nabla g(x^{(k)}) \) were costless, this method would be faster than Newton’s method when the number of dimensions \( n \) is large.