Math 484: Nonlinear Programming[1] Mikhail Lavrov

# Chapter 3, Lecture 6: Broyden's Method

*April 26, 2019* *University of Illinois at Urbana-Champaign*

## 1 Motivation

Previously, we assumed that computing the gradient $\nabla f(\mathbf{x})$ or the Hessian matrix $Hf(\mathbf{x})$ 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 $Hf(\mathbf{x})$ 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).

In one dimension, we discussed the secant method as a solution to the second of these problems. Here, we used the values at the last two points to estimate the derivative.

Now let's pass to the $n$-dimensional case: finding the solution to a system of equations $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ for a function $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$. To approximate Newton's method, we need an estimate of the Jacobian $\nabla \mathbf{g}$; in other words, a linear approximation of $\mathbf{g}$.

We encounter a problem: just the values of $\mathbf{g}$ at the last two points are not enough to build a complete linear approximation of $\mathbf{g}$. (In general, such an approximation would need at least $n + 1$ points to define.)

## 2 Rank one updates

Our solution to this problem 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.

Intuitively, when comparing the new value $\mathbf{g}(\mathbf{x}^{(k+1)})$ to the previous value $\mathbf{g}(\mathbf{x}^{(k)})$, we can learn something about the rate of change of $\mathbf{g}$ in the direction of $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, but nothing about the rate of change of $\mathbf{g}$ in other directions.

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 $\mathbf{x}^{(k)}$, we make a linear approximation to $\mathbf{g}$:

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

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

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

[1] This document comes from the Math 484 course webpage: https://faculty.math.illinois.edu/~mlavrov/courses/484-spring-2019.html

method, we make a linear approximation

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

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

In practice, $\mathbf{g}(\mathbf{x}^{(k+1)})$ doesn't end up being $\mathbf{0}$: if everything is going well, it should be closer to $\mathbf{0}$ than $\mathbf{g}(\mathbf{x}^{(k)})$ 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 $\mathbf{g}(\mathbf{x}^{(k+1)})$, 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

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

where we previously had

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

Second, we ask that in every direction orthogonal to the direction $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, 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

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

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 $\mathbf{b}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, then by taking the difference of (1) and (2), we get

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

while (3) can be rewritten as

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

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

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

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

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

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

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

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

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 $\mathbf{g}(\mathbf{x}^{(k+1)})$. (And its rows are all multiples of $(\mathbf{b}^{(k)})^\mathsf{T}$.) 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 $\mathbf{x}^{(0)}$; we also need a matrix $D_0$ to start as our approximation to $\nabla\mathbf{g}(\mathbf{x}^{(0)})$. (If computing derivatives of $\mathbf{g}$ 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 $\mathbf{x}^{(k+1)}$ and $D_{k+1}$ from $\mathbf{x}^{(k)}$ and $D_k$, we:

1. Solve $\mathbf{g}(\mathbf{x}^{(k)}) + D_k(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{0}$ for $\mathbf{x}^{(k+1)}$; equivalently, set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - D_k^{-1}\mathbf{g}(\mathbf{x}^{(k)})$.
2. Set $D_{k+1} = D_k + \frac{\mathbf{g}(\mathbf{x}^{(k+1)})(\mathbf{b}^{(k)})^\mathsf{T}}{\mathbf{b}^{(k)} \cdot \mathbf{b}^{(k)}}$, where $\mathbf{b}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$.

## 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

$$\begin{cases} x + y = 2, \\ x - y = 0 \end{cases}$$

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 $\mathbf{g}$ is $\mathbf{g}(x, y) = (x + y - 2, x - y)$.

Our first step sets

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

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

$$D_1 = D_0 + \frac{\mathbf{g}(x_1, y_1)(\mathbf{b}^{(0)})^\mathsf{T}}{\mathbf{b}^{(0)} \cdot \mathbf{b}^{(0)}}$$

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

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

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

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

Now, with information at more points, $D_2$ becomes the correct Jacobian matrix of the linear function $\mathbf{g}$. In the next step, our linear approximation to $\mathbf{g}$ will actually be $\mathbf{g}$, 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 $\nabla \mathbf{g}(\mathbf{x}^{(k)})$ after many steps. It is often the case that after enough steps, the matrix $D_k$ will accurately tell us what $\nabla \mathbf{g}(\mathbf{x}^{(k)})$ 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 $\nabla \mathbf{g}(\mathbf{x}^{(k)})^{-1}$; instead, we preferred to solve a system of linear equations at each step. With Broyden's method, however, the iterative step

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - D_k^{-1} \mathbf{g}(\mathbf{x}^{(k)})$$

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 $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, then $(A + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1}$ can be computed from $A^{-1}$ as

$$(A + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}A^{-1}}{1 + \mathbf{v}^\mathsf{T}A^{-1}\mathbf{u}}.$$

We can use this to compute $D_{k+1}^{-1}$ from $D_k^{-1}$ if we set $A = D_k$, $\mathbf{u} = \mathbf{g}(\mathbf{x}^{(k+1)})$, and $\mathbf{v} = \frac{\mathbf{b}^{(k)}}{\mathbf{b}^{(k)} \cdot \mathbf{b}^{(k)}}$. Since the Sherman–Morrison formula never requires multiplying two matrices together (we can write $A^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}A^{-1}$ as $A^{-1}\mathbf{u}$ times $\mathbf{v}^\mathsf{T}A^{-1}$), 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 $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - D_k^{-1}\mathbf{g}(\mathbf{x}^{(k)})$.
2. In terms of the vectors $\mathbf{b}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = -D_k^{-1}\mathbf{g}(\mathbf{x}^{(k)})$ and $\mathbf{c}^{(k)} = D_k^{-1}\mathbf{g}(\mathbf{x}^{(k+1)})$, set

$$D_{k+1}^{-1} = D_k^{-1} - \frac{\mathbf{c}^{(k)}\left((\mathbf{b}^{(k)})^{\mathsf{T}} D_k^{-1}\right)}{\mathbf{b}^{(k)} \cdot (\mathbf{b}^{(k)} + \mathbf{c}^{(k)})}.$$

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