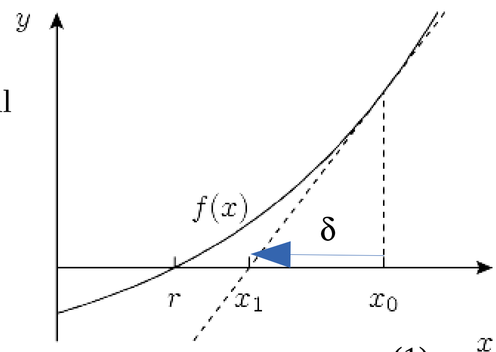
Derivation of Broyden's method

SDB, prepared for 2019 NA1 sessions.

Review of Newton's Method in N dimensions

Broyden's method is a replacement for Newton's method for solving systems of nonlinear equations. In Newton's method we assume we have a system of *N* nonlinear functions of *N* variables, $\vec{f}(\vec{x}): \mathbb{R}^N \rightarrow \mathbb{R}^N$. We want to find the roots of $\vec{f}$, i.e. the set of $\vec{x}$ for which $\vec{f}(\vec{x})=0$.

Newton's method is derived as follows. Assume we sit at an arbitrary point in our search space $\vec{x}_k$. (Here, the index *k* indicates that we will soon create an iterative method which will step towards the solution.) We wish to find the point where $\vec{f}(\vec{x})=0$. Imagine we could take a single step δ_k which would bring us to the root *r*. If our distance to the root is "sufficiently small", we can expand the function at our current point $\vec{x}_k$ to the new point $\vec{x}_k+\vec{\delta}_k$ using a Taylor's series



$$\vec{f}(\vec{x}_k+\vec{\delta}_k)=\vec{f}(\vec{x}_k)+J(\vec{x}_k)\vec{\delta}_k+\cdots \tag{1}$$

where $J(\vec{x}_k)$ is the Jacobian of $\vec{f}(\vec{x}_k)$ evaluated at the point $\vec{x}_k$.

Now if our step δ_k took us to exactly the correct position of the root, we would have

$$\vec{f}(\vec{x}_k+\vec{\delta}_k)=\vec{0}=\vec{f}(\vec{x}_k)+J(\vec{x}_k)\vec{\delta}_k+\cdots \tag{2}$$

Next, we make the assumption that truncating the Taylor's series after the linear term will provide a reasonable approximation to the desired step. This yields

$$\vec{\delta}_k=-J^{-1}(\vec{x}_k)\vec{f}(\vec{x}_k) \tag{3}$$

This gives a recipe for an iterative algorithm which will take repeated steps towards the actual root. If we take the next point in the sequence of steps to be $\vec{x}_{k+1}=\vec{x}_k+\vec{\delta}_k$, then the iteration looks like this:

$$\vec{x}_{k+1}=\vec{x}_k-J^{-1}(\vec{x}_k)\vec{f}(\vec{x}_k) \tag{4}$$

This is the iteration used by Newton's method. It is frequently shorted to read like this:

$$\vec{x}_{k+1}=\vec{x}_k-J_k^{-1}\vec{f}_k$$

Some drawbacks to Newton's method:

1. You need to figure out the analytic form of the Jacobian and then code the relevant expressions into your program. If you have a large system of equations, doing this by hand is tiresome and error-prone. Also, the number of elements in the Jacobian grows as N^2, which is fast growth if you think about doing hand calculations. For example, if you have a system of 10 equations,

the Jacobian will have 100 elements – think about doing 100 derivatives by hand!
2. The computer must evaluate a new Jacobian at every step of the method, which also represents an O(N^2) computational burden on the computer.
3. In some cases, a finite difference Jacobian is used instead of an analytic Jacobian. Although this eliminates the dreaded hand calculation, it still imposes an O(N^2) computational penalty on the computation.

Broyden's method

Broyden's method replaces the Jacobian with an approximation. The approximation is to use a matrix, *B*, which is updated at every step. The hope is that the update rule is easy to compute, and in particular does not require lots of hand calculations.

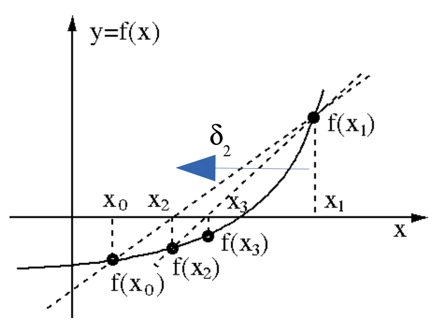
Here is a description of the thought process lying underlying Broyden's method. In what follows I will drop the arrows indicating vector quantities since at this point you know what is a vector and what is not.

Broyden's Method Algorithm:

1. Start at position x_k . Evaluate $f_k = f(x_k)$ at this point.
2. Compute the approximate Jacobian, B_k using $f_k, f_{k-1}, x_k, x_{k-1}$ and B_{k-1} . The method to compute B_k is detailed below. (At the beginning of the method -- i.e. the *k*=0 step -- one can use a finite difference Jacobian or some other approximation.)
3. Use the approximate Jacobian B_k to compute the step δ_{k+1} . Similar to equation (3) we use

$$\delta_{k+1} \leftarrow -B_k^{-1} f_k$$

4. Take the step to the new position, $x_{k+1} \leftarrow x_k + \delta_{k+1}$
5. Check for convergence, and if not converged go back to step 1.



Now the question is, how to update B_k ? If $f(x)$ was a scalar function of scalar x we could use the secant method which says that the new B_k is just a slope,

$$B_k = \frac{f_k - f_{k-1}}{x_k - x_{k-1}} \tag{5}$$

In these variables, the next step in the secant method is

$$\delta_{k+1} = -f_k / B_k$$

Refer to the figure for a reminder of how this works (and compare this to scalar Newton's method). However, in (5) neither *f* nor *x* are scalar variables – we are dealing with a system of equations, so these quantities are vectors. Therefore, one can't simply do the simple-minded division shown in (5).

So how to get B_k ? What if we express the information embedded in (5) in a matrix-friendly way, like this:

$$f_k - f_{k-1} = B_k (x_k - x_{k-1}) \tag{6}$$

Rewrite it as

$$\Delta_k = B_k \delta_k \tag{7}$$

where

$$\Delta_k = f_k - f_{k-1}$$

and

$$\delta_k = x_k - x_{k-1}$$

Equation (6) has the following problem: We know *f* and *x*. They contain *N* known quantities. However, the unknown matrix *B* contains N^2 elements. The problem is underdetermined. Therefore, to get B_k from the quantities we know, we need to impose additional constraints. Broyden's method imposes the following constraints:

1. We insist that updates to the matrix *B* are rank 1,

$$B_k - B_{k-1} = \text{ rank 1 } = u v^T \tag{8}$$

   where *u*, *v* are some vectors to be determined.

2. We insist the update minimizes the Frobenius norm of the difference:

$$minimize \|B_k - B_{k-1}\|_F$$

   where the Frobenius norm is defined as

$$\|B\|_F = \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} |b_{ij}^2|} = \sqrt{Tr(B B^T)} = \sqrt{Tr(B^T B)}$$

   and *Tr*(*B*) designates the trace of the matrix *B*.

Now consider constraint 1, equation (8). Write it as

$$B_k = B_{k-1} + u v^T \tag{9}$$

Multiply through on the right by δ_k to get

$$B_k \delta_k = B_{k-1} \delta_k + u (v^T \delta_k) \tag{10}$$

Note that I have grouped v^T and δ_k together, so the combination is a dot product. As long as $v^T \delta_k \neq 0$ we can manipulate (10) using equation (7) to obtain an expression for *u*,

$$u = \left(\frac{\Delta_k - B_{k-1} \delta_k}{v^T \delta_k} \right)$$

Inserting this back into (9), we get

$$B_k = B_{k-1} + \left(\frac{\Delta_k - B_{k-1} \delta_k}{v^T \delta_k} \right) v^T \tag{11}$$

This expression is valid for all $v^T \delta_k \neq 0$. Therefore, it is valid if we choose $v^T = \delta_k^T$. Inserting this expression into (11), we get Broyden's method to update the matrix *B*,

$$B_k = B_{k-1} + (\Delta_k - B_{k-1}\delta_k)\left(\frac{\delta_k^T}{\delta_k^T \delta_k}\right) \tag{12}$$

This expression is Broyden's method.

Comments on Broyden's method:

First off, it's clear the update is rank one because the right hand term is an outer product of two vectors. Therefore, constraint 1 is obeyed.

What about constraint 2? I claim that making the choice $v = \delta_k$ satisfies the condition that (12) minimizes the norm. Here's the argument:

Consider the constrained minimization problem:

$$\text{minimize } \|B_k - B_{k-1}\|_F \text{ subject to } \Delta_k = B_k \delta_k$$

From (11) we have

$$B_k - B_{k-1} = (\Delta_k - B_{k-1}\delta_k)\left(\frac{v^T}{v^T \delta_k}\right)$$

So taking the Frobenius norm

$$\|B_k - B_{k-1}\|_F = \left\| (\Delta_k - B_{k-1}\delta_k)\frac{v^T}{v^T \delta_k} \right\|_F$$

Now use $\Delta_k = B_k \delta_k$ to get

$$\|B_k - B_{k-1}\|_F = \left\| (B_k - B_{k-1})\frac{\delta_k v^T}{v^T \delta_k} \right\|_F$$

Using the triangle inequality, we get

$$\|B_k - B_{k-1}\|_F \le \|B_k - B_{k-1}\|_F \left\| \frac{\delta_k v^T}{v^T \delta_k} \right\|_F$$

or

$$1 \le \left\| \frac{\delta_k v^T}{v^T \delta_k} \right\|_F = \frac{\|\delta_k v^T\|_F}{v^T \delta_k} \tag{13}$$

Now consider the Frobenius norm of $\delta_k v^T$. By definition it is

$$\|\delta_k v^T\|_F = \sqrt{Tr(v \delta_k^T \delta_k v^T)}$$

$$=\sqrt{(\delta_k^T \delta_k) Tr(v v^T)}$$

$$=\sqrt{(\delta_k^T \delta_k) Tr(v^T v)}$$

$$=\sqrt{(\delta_k^T \delta_k)(v^T v)}$$

Inserting this expression back into (13) gives

$$1 \leq \frac{\sqrt{\delta_k^T \delta_k v^T v}}{v^T \delta_k}$$

When $\delta_k = v$ this is an equality, meaning the this is the value which minimizes the expression (13), as was to be shown. Therefore, the update rule (12) satisfies both constraints.