APPM 4650 — Quasi-Newton methods

Warm-up:

(a) What is Newton's method for solving the following system of equations?

$$\begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(b) What is the most expensive part of creating the new iterate?

Soln:

(a)

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - (\mathsf{G}(\boldsymbol{x}_n))^{-1} \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix}$$

where

$$\mathsf{G}(\boldsymbol{x}_n) = \begin{bmatrix} f_x(x_n, y_n) & f_y(x_n, y_n) \\ g_x(x_n, y_n) & g_y(x_n, y_n) \end{bmatrix}$$

and $\boldsymbol{x}_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$.

(b) The most expensive part is creating the inverse of the Jacobian at each step. This is even more trouble when we are trying to solve a large system of equations.

Today we will talk about techniques that avoid having to invert the Jacobian at each step but still have decent convergence properties. These methods are called *Quasi-Newton methods*. In some way, you can think of the secant method as a quasi-Newton method as it avoids having to deal with evaluating the derivative.
Our goal in creating a quasi-Newton method for systems of equations is to avoid inverting a new matrix at each step. With this decrease in computational cost, we will experience a sacrifice in the order of the method. While Newton's method is (often) second order convergent, quasi-Newton methods are between first and second order (*super linear*).

Option 1: Lazy Newton The idea behind Lazy Newton is to invert the Jacobian once and use it for all iterations. In other words, let

$$\hat{\mathsf{G}} = \mathsf{G}(\boldsymbol{x}_0) \text{ and } \hat{\mathsf{G}}^{-1} = (\mathsf{G}(\boldsymbol{x}_0))^{-1}.$$

Then our iteration becomes

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \hat{\mathsf{G}}^{-1} \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix}.$$

This will suffer and may not converge if our initial guess is not close enough to the root at the beginning; i.e. not in the basin of convergence. On the upside the cost of creating a new iterate is $O(n^2)$ instead of $O(n^3)$ of the real Newton's method.

Draw a picture of what this looks like for the scalar function problem.

Option 2: Broyden method This is a rough extension of the Secant method.
Recall that for Secant we approximated the derivative at each iteration with the slope of a secant line.
The idea for Broyden method is that given $\mathsf{G}(\boldsymbol{x}_0)$ can we write $\mathsf{G}(\boldsymbol{x}_1)$ as a rank 1 update to $\mathsf{G}(\boldsymbol{x}_0)$.

Note: a rank one matrix can be written as the outer product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. In other words, the matrix $\boldsymbol{x}\boldsymbol{y}^T$ is a rank one matrix.

There are two glaring questions:

1. Why is this a viable idea?
2. How do we find the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ to create the rank 1 update?

Theorem 0.1 (Sherman-Morrison Formula (Theorem 10.8))**.** *Given* A*,* A^{-1} *and a rank one update* $\boldsymbol{x}\boldsymbol{y}^T$*, the inverse of* $\mathsf{A}+\boldsymbol{x}\boldsymbol{y}^T$ *is given as follows:*

$$\left(\mathsf{A}+\boldsymbol{x}\boldsymbol{y}^T\right)^{-1} = \mathsf{A}^{-1} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}}{1+\boldsymbol{y}\mathsf{A}^{-1}\boldsymbol{x}}$$

Note: The inverse of the matrix $\mathsf{A}+\boldsymbol{x}\boldsymbol{y}^T$ can be applied using only A^{-1} and inner products. Exploiting this means that the inverse can be applied to a vector for a cost of $O(n^2)$.

Proof. We will only prove one direction. You can do the other direction on your own.

$$\begin{aligned}\left(\mathsf{A}^{-1} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}}{1+\boldsymbol{y}\mathsf{A}^{-1}\boldsymbol{x}}\right)(\mathsf{A}+\boldsymbol{x}\boldsymbol{y}^T) &= \mathsf{A}^{-1}\mathsf{A} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}\mathsf{A}}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}} + \mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}} \\ &= \mathsf{I} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}} + \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T(1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x})}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}} \\ &= \mathsf{I} + \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x} - \mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{y}^T}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}}\end{aligned}$$

Let $c = \boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}$. Then

$$\begin{aligned}\left(\mathsf{A}^{-1} - \frac{\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\mathsf{A}^{-1}}{1+\boldsymbol{y}\mathsf{A}^{-1}\boldsymbol{x}}\right)(\mathsf{A}+\boldsymbol{x}\boldsymbol{y}^T) &= \mathsf{I} + \frac{c}{1+\boldsymbol{y}^T\mathsf{A}^{-1}\boldsymbol{x}}\left(\mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T - \mathsf{A}^{-1}\boldsymbol{x}\boldsymbol{y}^T\right) \\ &= \mathsf{I}\end{aligned}$$

$\square$

Now we need to figure out what the update to our matrix is.
Let $\boldsymbol{F}(\boldsymbol{x}_n) = \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n)\end{bmatrix}$.
The we will define our approximate Jacobian at iteration $n+1$ by

$$\hat{\mathsf{G}}_{n+1} = \hat{\mathsf{G}}_n + \frac{\left[\boldsymbol{F}(\boldsymbol{x}_{n+1}) - \boldsymbol{F}(\boldsymbol{x}_n) - \hat{\mathsf{G}}_n(\boldsymbol{x}_{n+1}-\boldsymbol{x}_n\right](\boldsymbol{x}_{n+1}-\boldsymbol{x}_n)^T}{\|\boldsymbol{x}_{n+1}-\boldsymbol{x}_n\|_2^2}$$

So our vectors are $\boldsymbol{x} = \left[\boldsymbol{F}(\boldsymbol{x}_{n+1}) - \boldsymbol{F}(\boldsymbol{x}_n) - \hat{\mathsf{G}}_n(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\right]$ and $\boldsymbol{y}^T = \frac{(\boldsymbol{x}_{n+1}-\boldsymbol{x}_n)^T}{\|\boldsymbol{x}_{n+1}-\boldsymbol{x}_n\|_2^2}$.

Numerical example: Consider the task of trying to approximate the roots of the nonlinear system of equations

$$\begin{cases} 3x - \cos(yz) - 1/2 & = 0 \\ x - 81(y+0.1)^2 + \sin(z) + 1.06 & = 0 \\ e^{-xy} + 20z + \frac{10\pi - 3}{3} & = 0 \end{cases}$$

with initial guess $\boldsymbol{x}_0 = \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix}$.

How long does this take to converge for Newton's method versus Broyden's method?