More quasi-Newton

In last class, we saw that in each step of a quasi-Newton, we update $B_{k+1} = B_k + C_k$, where C_k is low rank (in general, the rank of C_k is 1 or 2 — DFS and BFGS, respectively). So we can write, e.g.:

$$C_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T.$$

In general, a rank p matrix $A \in \mathbb{R}^{m \times n}$ can always be written as a p-term sum of outer products using the singular value decomposition (SVD):

$$A = U \Sigma V^T = \underbrace{\begin{bmatrix} u_1 & \cdots & u_p \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_p \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} v_1 & \cdots & v_p \end{bmatrix}^T}_{V}.$$

Technically, this is the "reduced" SVD of A. In this class, we will not need to make use of the SVD.

The problem with this iteration for B_k is that we still need its inverse. Let's assume B_k is symmetric and positive definite as we iterate. A symmetric positive definite matrix A admits a Cholesky decomposition:

$$A = LL^T,$$

where L is lower triangular. It can be obtained by using Gaussian elimination in matrix form to row reduce A.

The cost of computing the Cholesky decomposition is $O(n^3)$ if $A \in \mathbb{R}^{n \times n}$. Afterwards, we can solve the system $Ax = b$ using a "forward solve" and a "backward solve" in $O(n^2)$ time.

The problem with our general quasi-Newton iteration as we have formulated it is that to find p_k, we still need to solve:

$$B_k p_k = -\nabla f(x_k), \quad k = 0, 1, \ldots$$

requiring potentially $O(n^3)$ work per step. This is no better than Newton's method!

Let's look at two ways to improve this situation. First, let's see how to "update" a Cholesky factorization after making a rank-1 modification to it.

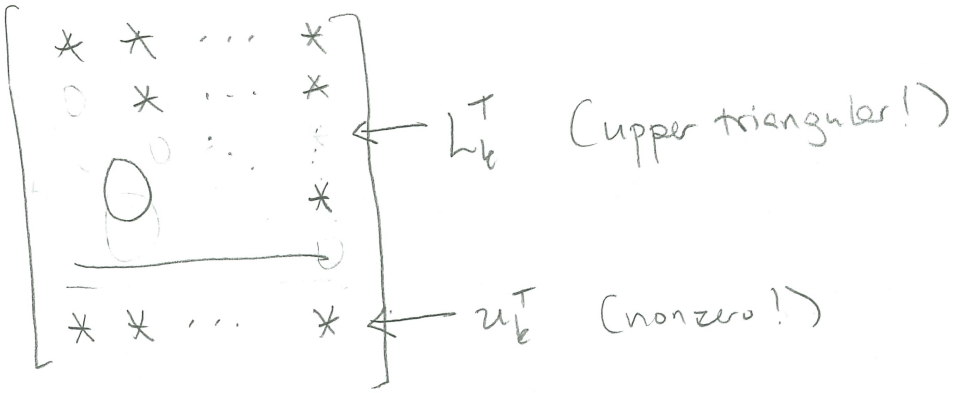
First, let $B_k = L_k L_k^T$ be the Cholesky decomposition of B_k, and assume $C_k = u_k u_k^T$, so that: $B_{k+1} = B$

$$B_{k+1} = B_k + u_k u_k^T = L_k L_k^T + u_k u_k^T.$$

This can be rewritten (check it) as:

$$B_{k+1} = \begin{bmatrix} L_k & u_k \end{bmatrix} \begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix} = \begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix}^T \begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix}.$$

The zero pattern of $\begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix}$ is:



This means we need to zero out the final row of $\begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix}$.

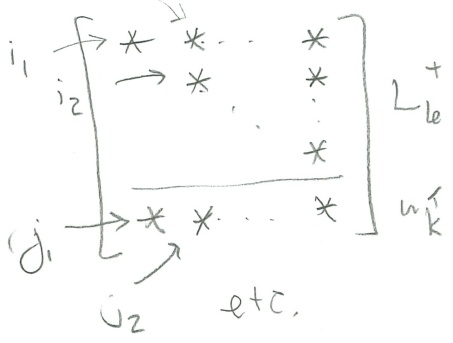
A standard way of doing this is using Givens rotations.

These are matrices of the form:

$$G_{ij}(\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \cos(\theta) & & -\sin(\theta) & & \\ & & & \ddots & & & \\ & & \sin(\theta) & & \cos(\theta) & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow i\text{th row} \\ \\ \leftarrow j\text{th row} \\ \\ \\ \end{matrix}$$

$$\uparrow \quad i\text{th col.} \qquad \uparrow \quad j^{\text{th}}\text{ col.}$$

Then, i, j, and θ can be chosen to zero out a single element. The idea is as follows:



Construct n Givens rotations to zero out the nonzero entries of $\begin{bmatrix} L_k^T \\ u_k^T \end{bmatrix}$ corresponding to u_k^T. This changes L_k^T!

Now, let's multiply all the Givens rotations together to get a matrix Q:

$$Q = G_n \cdots G_1.$$

Note that since each G_i is a rotation matrix it is an orthogonal matrix...

finish...

Another approach uses the Sherman-Morrison formula:

Theorem: (Sherman-Morrison)

Let $A \in \mathbb{R}^{n \times n}$ be invertible. Then, $A + uv^T$ is invertible iff $1 + v^T A^{-1} u \neq 0$. Then:

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

Proof: see internet.

Let's apply this to the SR1 update:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

We get:

$$B_{k+1}^{-1} = \left(B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \right)^{-1}$$

$$= B_k^{-1} - \frac{B_k^{-1} \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} B_k^{-1}}{1 + \frac{(y_k - B_k s_k)^T B_k^{-1} (y_k - B_k s_k)}{(y_k - B_k s_k)^T s_k}},$$

where $u = \frac{y_k - B_k s_k}{(y_k - B_k s_k)^T s_k}$ and $v = y_k - B_k s_k$.

If we can compute B_0^{-1} in $O(n^2)$ time, then this gives us an $O(n^2)$ update of the inverse.

Can this formula be simplified?