Fast log-decon with a quasi-Newton solver

Antoine Guitton

ABSTRACT

I speed up the log-decon method by replacing the slow steepest-descent method with a faster quasi-Newton technique known as the limited-memory BFGS.

INTRODUCTION

The log-decon method of Claerbout et al. (2011) estimates a filter that can both handle non-minimum phase wavelets (e.g., Ricker) and produce sparse seismic reflections where the polarity is easily identifiable. This method is extended in Claerbout et al. (2012) to include variable gain. Claerbout proposes to compute the filter coefficients with a steepest-descent approach, where the step length can be estimated very accurately with a Newton-search technique. Steepest descent is notoriously slow: its convergence rate depends on the conditioning of the problem producing a well-known zig-zagging effect close to the solution. Here, I propose to employ the L-BFGS method, a quasi-Newton technique that improves the convergence and decreases the number of iterations.

TWO SOLVERS

In this section, I follow Claerbout's notations for all the variables: lower case letters are for variables in time and space, while upper case letters are for variables in frequency and space.

The slow steepest-descent method

The steepest descent method requires the computation of the gradient. The model space is a vector of filter coefficients $u(t)$. Claerbout shows that the gradient $du(t)$ of the sparse log-decon method corresponds to the crosscorrelation of the residual (the reflectivity series) with the soft-clipped residual (see Claerbout et al. (2012) for a generalization with a variable gain). The pseudo-code below shows the steepest descent algorithm.

Once $u(t)$ is estimated, we obtain the wavelet $w(t) = FT^{-1}\left(e^{-U(\omega)}\right)$ and the sparse decon output $r(t) = FT^{-1}\left(D(\omega)e^{U(\omega)}\right)$, where $D(\omega)$ is the Fourier transformed input data.

The fast L-BFGS method

The L-BFGS method is a member of the quasi-Newton family: it updates at each iteration an approximation of the inverse Hessian **Q**. This technique is very cost effective: given the most recent history of gradient and model vectors (usually around 5) kept in memory, the quasi-Newton search direction **Qdu** (inverse Hessian times the gradient) is computed directly with simple vector multiplications. Therefore, the L-BFGS solver can be used for large non-linear problems (Nocedal, 1980).

Contrary to steepest descent where the step length is estimated with a Newton-search technique, the step length in L-BFGS is computed such that sufficient decrease of the error and of the local curvature is attained (so called "Wolfe conditions"). The appendix shows the L-BFGS solver in more details. The L-BFGS code can be downloaded at http://users.eecs.northwestern.edu/~nocedal/lbfgs.html The pseudo-code below shows both the steepest-descent and L-BFGS algorithms.

```
U = 0.        # or other initializations
Remove the mean from U(omega).

Iteration {
     dU = 0
     Compute dU
     Remove mean from dU
     du = FT(dU)
     if (steepest descent)
       {
       Compute alfa with Newton iterations
       u  = u + alfa*du
       }
     else if (L-BFGS)
       {
       Compute alfa with More and Thuente method
       u  = u + alfa*Qdu # Q = inverse approximate Hessian
       }
          }
```

A fair warning

Comparing the convergence of optimization techniques can be quite difficult to do in a fair manner. My steepest descent algorithm includes a termination criterion based on the Armijo rule only, namely, a sufficient decrease condition of the objective function (whereas L-BFGS use the Wolfe conditions). In addition, both the steepest descent and L-BFGS algorithms have different line-searches, which will affect convergence and computational performances. The L-BFGS line-search is based on the More and Thuente method, which uses bracketing and quadratic/cubic interpolation. The steepest-descent algorithm uses a very simple scheme where the step length is divided by two until the sufficient decrease condition is respected. Therefore, some of the computational differences come from the line-search algorithm and stopping criteria in addition to the inherent convergence properties of the two methods.

A COMPARISON ON A FIELD-DATA EXAMPLE

Figure 1 shows a near-offset section of a 2-D line from the Gulf of California used in Claerbout et al. (2012). The left side displays the input data and the right side the deconed data when the L-BFGS solver is used. I do not show the result of the steepest-descent because both methods estimate very similar wavelets, as shown in Figure 2. As expected, the reflectivity of the deconed data in Figure 1 is revealed quite well: the polarity of large reflectors is enhanced. Also, a non-minimum phase wavelet is obtained, regardless of the method (Figure 2).

In terms of convergence speed, it takes 35 iterations and 1.7 seconds for the L-BFGS technique to reach a minimum, and 123 iterations and 24 seconds for the steepest-descent algorithm (Figure 3). Ignoring the time it takes to read and write data on disk, each iteration of the L-BFGS algorithm is about five times faster, with almost four times less iterations. Clearly, this difference is not solely due to the better convergence properties of the quasi-Newton algorithm over the steepest-descent method. As mentioned before, different line-search strategies and stopping criteria matter as well.

CONCLUSION

The log-decon method is sped up by using the L-BFGS method compared to the steepest-descent solver. A factor 20 is observed in this paper but results will vary depending on the experimental setup.

APPENDIX

The L-BFGS method is suitable for smooth functions where local minima exist. It is not a method for global optimization where the global minimum is sought. L-BFGS is presented here in general terms using global definitions for the different variables: it does not follow the notations of the log-decon method.

We define $\mathbf{m}^*$ a local minimizer for $f(\mathbf{m})$ and we assume that $f(\mathbf{m})$ and $\mathbf{m}^*$ satisfy the "standard requirements":

1. f is twice differentiable,

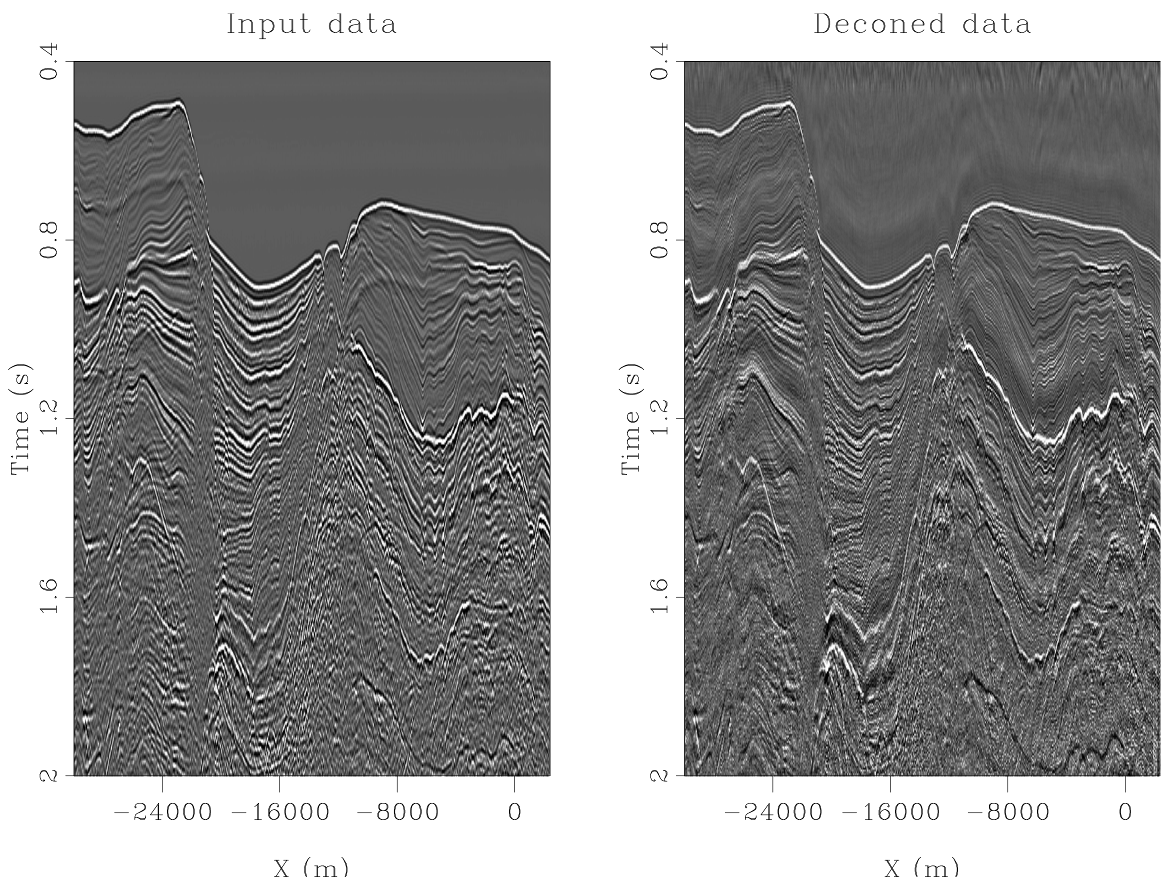

Figure 1: Left: input data. Right: deconed data with the L-BFGS solver

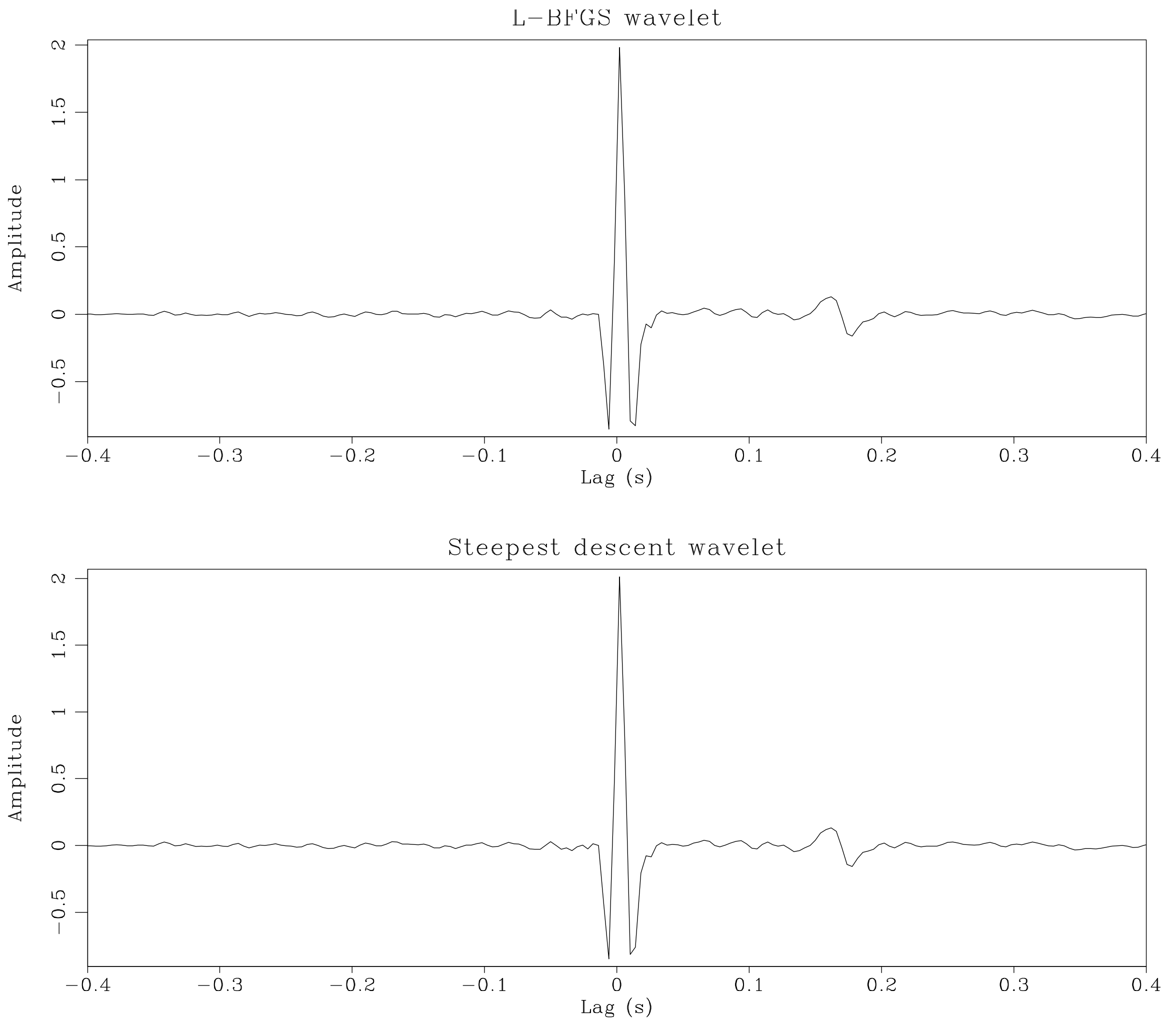

Figure 2: Top: wavelet estimated with the L-BFGS method. Bottom: wavelet estimated with the steepest-descent method.

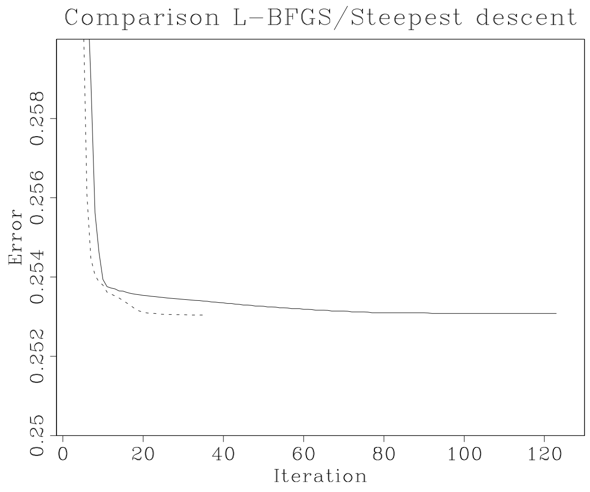

Figure 3: Convergence comparison between the steepest descent (solid) and L-BFGS (dash) methods. Note the narrow horizontal scale to highlight small differences close to the convergence point.

2. $\nabla f(\mathbf{m}^*) = 0$,
3. $\nabla^2 f(\mathbf{m}^*)$ is positive definite , i.e., $\mathbf{m}'\nabla^2 f(\mathbf{m}^*)\mathbf{m} > 0$ for all $\mathbf{m} \in \Re^N$ ($'$ denotes the adjoint).

where N is the dimension of the model vector $\mathbf{m}$ and $\Re^N$ the real space for the model vector $\mathbf{m}$. Any vector $\mathbf{m}^*$ that satisfies the standard requirements is a local minimizer of $f(\mathbf{m})$.

Newton's method is an iterative process where the solution to the problem is updated as follows:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \lambda_k \mathbf{H}_k^{-1} \nabla f(\mathbf{m}_k), \tag{1}$$

where $\mathbf{m}_{k+1}$ is the updated solution at iteration $k+1$, λ_k the step length computed by a line search that ensures a sufficient decrease of $f(\mathbf{m})$ and $\mathbf{H}_k = \nabla^2 f(\mathbf{m}_k)$ the Hessian (or second derivative). In many circumstances the inverse of the Hessian can't be computed directly. It happens for example when the matrix $\mathbf{H}$ is too big or when operators are used rather than matrices. Fortunately we might be able to compute an approximation of the Hessian of $f(\mathbf{m})$. This strategy gives birth to quasi-Newton methods where the way in which the Hessian is computed determines the method (Kelley, 1999).

A possible update of the Hessian is given by the BFGS technique Broyden (1969); Fletcher (1970); Goldfarb (1970); Shanno (1970). The BFGS update is given by

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}\mathbf{y}'}{\mathbf{y}'\mathbf{s}} - \frac{(\mathbf{H}_k\mathbf{s})(\mathbf{H}_k\mathbf{s})'}{\mathbf{s}'\mathbf{H}_k\mathbf{s}}, \tag{2}$$

where $\mathbf{s} = \mathbf{m}_{k+1} - \mathbf{m}_k$ and $\mathbf{y} = \nabla f(\mathbf{m}_{k+1}) - \nabla f(\mathbf{m}_k)$. In practice, however, we rather write the previous equation in terms of the inverse matrices. We have then

$$\mathbf{H}_{k+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{s}\mathbf{y}'}{\mathbf{y}'\mathbf{s}}\right) \mathbf{H}_k^{-1} \left(\mathbf{I} - \frac{\mathbf{y}\mathbf{s}'}{\mathbf{y}'\mathbf{s}}\right) + \frac{\mathbf{s}\mathbf{s}'}{\mathbf{y}'\mathbf{s}}. \tag{3}$$

In addition, we use the history of the iterations to compute the new Hessian rather than a full storage of the matrix $\mathbf{H}_k^{-1}$. This requires that a gradient step vector $\mathbf{y}$ and a solution step vector $\mathbf{s}$ are kept in memory after each iteration. Consequently this method might not been affordable with large data and model space. In the next section a modified version of the BFGS method that limits the storage needed to compute the update of the Hessian is proposed.

The limited-memory BFGS method

Nocedal (1980) derives a technique that partially solves the storage problem caused by the BFGS update. Instead of keeping all the $\mathbf{s}$ and $\mathbf{y}$ from the past iterations, we update the Hessian using the information from the l previous iterations, where l is given by the end-user. This implies that when the number of iterations is smaller than l, we have the usual BFGS update, and when it is larger than l, we have a limited-memory BFGS (L-BFGS) update.

I give the updating formulas of the Hessian as presented by Nocedal (1980). First, we define

$$\rho_i = 1/\mathbf{y}_i'\mathbf{s}_i,\ \mathbf{v}_i = (I - \rho_i \mathbf{y}_i \mathbf{s}_i') \text{ and } \mathbf{H}^{-1} = \mathbf{B}.$$

As described above, when k, the iteration number, obeys $k+1 \leq l$, where l is the storage limit, we have the BFGS update

$$\begin{aligned} \mathbf{B}_{k+1} &= \mathbf{v}_k'\mathbf{v}_{k-1}'\cdots\mathbf{v}_0'\mathbf{B}_0\mathbf{v}_0\cdots\mathbf{v}_{k-1}\mathbf{v}_k \\ &\quad +\mathbf{v}_k'\cdots\mathbf{v}_1'\rho_0\mathbf{s}_0\mathbf{s}_0'\mathbf{v}_1\cdots\mathbf{v}_k \\ &\quad \vdots \\ &\quad +\mathbf{v}_k'\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}'\mathbf{v}_k \\ &\quad +\rho_k\mathbf{s}_k\mathbf{s}_k'. \end{aligned} \tag{4}$$

For $k+1 > l$ we have the limited-memory update

$$\begin{aligned} \mathbf{B}_{k+1} &= \mathbf{v}_k'\mathbf{v}_{k-1}'\cdots\mathbf{v}_{k-l+1}'\mathbf{B}_0\mathbf{v}_{k-l+1}\cdots\mathbf{v}_{k-1}\mathbf{v}_k \\ &\quad +\mathbf{v}_k'\cdots\mathbf{v}_{k-l+2}'\rho_{k-l+1}\mathbf{s}_{k-l+1}\mathbf{s}_{k-l+1}'\mathbf{v}_{k-l+2}\cdots\mathbf{v}_k \\ &\quad \vdots \\ &\quad +\mathbf{v}_k'\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}'\mathbf{v}_k \\ &\quad +\rho_k\mathbf{s}_k\mathbf{s}_k'. \end{aligned} \tag{5}$$

These equations show how the update of the Hessian is calculated.

Usually the L-BFGS method is implemented with a line search for the step length λ_k to ensure a sufficient decrease of the misfit function. Convergence properties of the L-BFGS method are guaranteed if λ_k in equation (2) satisfies the *Wolfe conditions* (Kelley, 1999):

$$f(\mathbf{x}_k + \lambda_k\mathbf{d}_k) \leq f(\mathbf{x}_k) + \mu\lambda_k\nabla f(\mathbf{x}_k)'\mathbf{d}_k, \tag{6}$$

$$|\nabla f(\mathbf{x}_k + \lambda_k\mathbf{d}_k)'\mathbf{d}_k| \geq \nu|\nabla f(\mathbf{x}_k)'\mathbf{d}_k|. \tag{7}$$

ν and μ are constants to be chosen a priori and $\mathbf{d}_k = -\mathbf{B}_k\nabla f(\mathbf{m}_k)$. For ν and μ we set $\nu = 0.9$ and $\mu = 10^{-4}$ as proposed by Liu and Nocedal (1989). Equation (6) is a sufficient decrease condition that all line search algorithms must satisfy. Equation (7) is a curvature condition. The line search algorithm has to be carefully designed since it absorbs most of the computing time. I programmed a line search based on the More and Thuente (1994) method. Because the line search is time consuming, the step length $\lambda_k = 1$ is always tested first. This procedure saves a lot of computing time and is also recommended by Liu and Nocedal (1989). I now give the algorithm used to minimize any objective function involving nonlinear problems.

An efficient algorithm for solving nonlinear problems

The solver works as follows:

1. Choose $\mathbf{m}_0$, l, $\mathbf{B}_0$. Set $k = 0$.
2. Compute
$$\begin{aligned} \mathbf{d}_k &= -\mathbf{B}_k \nabla f(\mathbf{m}_k), & (8)\\ \mathbf{m}_{k+1} &= \mathbf{m}_k + \lambda_k \mathbf{d}_k, & (9) \end{aligned}$$
where λ_k meets the Wolfe conditions.
3. Let $\hat{l}$=min$\{k,\ l-1\}$. Update $\mathbf{B}_0$ $\hat{l}+1$ times using the pairs $\{\mathbf{y}_i, \mathbf{s}_i\}_{j=k-\hat{l}}^{k}$, i.e., let
$$\begin{aligned} \mathbf{B}_{k+1} = & \ \mathbf{v}_k'\mathbf{v}_{k-1}' \cdots \mathbf{v}_{k-\hat{l}}'\mathbf{B}_0\mathbf{v}_{k-\hat{l}} \cdots \mathbf{v}_{k-1}\mathbf{v}_k \\ & +\mathbf{v}_k' \cdots \mathbf{v}_{k-\hat{l}+1}'\rho_{k-\hat{l}}\mathbf{s}_{k-\hat{l}}\mathbf{s}_{k-\hat{l}}'\mathbf{v}_{k-\hat{l}+1} \cdots \mathbf{v}_k \\ & \vdots & (10) \\ & +\mathbf{v}_k'\rho_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}'\mathbf{v}_k \\ & +\rho_k\mathbf{s}_k\mathbf{s}_k'. & (11) \end{aligned}$$
4. Set $k = k+1$ and go to 2 if the residual power is not small enough.

The update $\mathbf{B}_{k+1}$ is not formed explicitly; instead we compute $\mathbf{d}_k = -\mathbf{B}_k \nabla f(\mathbf{x}_k)$ with an iterative formula Nocedal (1980). Liu and Nocedal (1989) propose scaling the initial symmetric positive definite $\mathbf{B}_0$ at each iteration as follows:

$$\mathbf{B}_k^0 = \frac{\mathbf{y}_k'\mathbf{s}_k}{\parallel \mathbf{y}_k \parallel_2^2}\mathbf{B}_0. \qquad (12)$$

This scaling greatly improves the performances of the method. Liu and Nocedal (1989) show that the storage limit for large-scale problems has little effects. A common choice for l is $l = 5$. In practice, the initial guess $\mathbf{B}_0$ for the Hessian is the identity matrix $\mathbf{I}$; then it might be scaled as proposed in equation (12). The nonlinear solver as detailed in the previous algorithm converges to a local minimizer $\mathbf{m}^*$ of $f(\mathbf{m})$.

REFERENCES

Broyden, C. G., 1969, A new double-rank minimization algorithm: AMS Notices, **16**, 670.

Claerbout, J., Q. Fu, and Y. Shen, 2011, A log spectral approach to bidirectional deconvolution: SEP-Report, **143**, 297–300.

Claerbout, J., A. Guitton, and Q. Fu, 2012, Decon in the log domain with variable gain: SEP-Report, **147**, 313–322.

Fletcher, R., 1970, A new approach to variable metric methods: Comput. J., **13**, 317–322.

Goldfarb, D., 1970, A family of variable metric methods derived by variational means: Math. Comp., **24**, 23–26.

Kelley, C. T., 1999, Iterative methods for optimization: SIAM in applied mathematics.

Liu, D. C. and J. Nocedal, 1989, On the limited memory BFGS method for large scale optimization: Mathematical Programming, **45**, 503–528.

More, J. J. and J. Thuente, 1994, Line search algorithms with guaranteed sufficient decrease: ACM Transactions on Mathematical Software, **20**, 286–307.

Nocedal, J., 1980, Updating quasi-Newton matrices with limited storage: Mathematics of Computation, **95**, 339–353.

Shanno, D. F., 1970, Conditioning of quasi-Newton methods for function minimization: Math. Comp., **24**, 647–657.