NORTHWESTERN UNIVERSITY
Department of Electrical Engineering
and Computer Science

REPRESENTATIONS OF QUASI-NEWTON MATRICES AND THEIR USE IN LIMITED MEMORY METHODS

by

Richard H. Byrd,[1] *Jorge Nocedal*[2] *and Robert B. Schnabel*[1]

Technical Report NAM-03

June 1992; revised January 21, 1996

[1] Computer Science Department, University of Colorado, Campus Box 430, Boulder, Colorado 80309. These authors were supported by the Air Force Office of Scientific Research under Grant AFOSR-90-0109, the Army Research Office under Grant DAAL03-91-0151 and the National Science Foundation under Grants CCR-8920519 and CCR-9101795

[2] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston Il 60208. This author was supported by the U.S. Department of Energy, under Grant DE-FG02-87ER25047-A001, and by National Science Foundation Grants CCR-9101359 and ASC-9213149 .

REPRESENTATIONS OF QUASI-NEWTON MATRICES AND THEIR USE IN LIMITED MEMORY METHODS

by

Richard H. Byrd, Jorge Nocedal and Robert B. Schnabel

ABSTRACT

We derive compact representations of BFGS and symmetric rank-one matrices for optimization. These representations allow us to efficiently implement limited memory methods for large constrained optimization problems. In particular, we discuss how to compute projections of limited memory matrices onto subspaces. We also present a compact representation of the matrices generated by Broyden's update for solving systems of nonlinear equations.

Key words: Quasi-Newton method, constrained optimization, limited memory method, large-scale optimization.

Abbreviated title: Representation of quasi-Newton matrices.

1. Introduction.

Limited memory quasi-Newton methods are known to be effective techniques for solving certain classes of large-scale unconstrained optimization problems (Buckley and Le Nir (1983), Liu and Nocedal (1989), Gilbert and Lemaréchal (1989)) . They make simple approximations of Hessian matrices, which are often good enough to provide a fast rate of linear convergence, and require minimal storage. For these reasons it is desirable to use limited memory approximations also for solving problems that include constraints. However, most algorithms for constrained optimization require the projection of Hessian approximations onto the subspace of active constraints and other matrix calculations that can be expensive when the number of variables is large. This is true even if limited memory approximations are used, unless special care is taken in their representation and manipulation.

In this paper we derive new representations of limited memory quasi-Newton matrices and show how to use them efficiently in the kind of matrix computations required in constrained optimization methods. We present new expressions for both the BFGS and symmetric rank-one formulae for optimization, and also derive a compact expression for Broyden's method for solving systems of nonlinear equations. We believe that these new compact representations of quasi-Newton matrices are of interest in their own right, but in this paper we focus on their use in limited memory methods.

To motivate the new matrix representations we begin by describing the limited memory BFGS method for unconstrained optimization. It is a variation of the standard BFGS method, which is given by

$$x_{k+1} = x_k - \lambda_k H_k g_k \quad k = 0, 1, 2, ... \tag{1.1}$$

where λ_k is a steplength, g_k is the gradient of the objective function $f : \mathbf{R}^n \to \mathbf{R}$ at x_k, and where the inverse Hessian approximation H_k is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{1.2}$$

where

$$\rho_k = 1/y_k^T s_k, \qquad V_k = I - \rho_k y_k s_k^T, \tag{1.3}$$

and

$$s_k = x_{k+1} - x_k, \qquad y_k = g_{k+1} - g_k.$$

(see e.g. Fletcher (1987)). We say that the matrix H_{k+1} is obtained by updating H_k using the pair $\{s_k, y_k\}$.

The limited memory BFGS method is an adaptation of the BFGS method to large problems. The implementation described by Liu and Nocedal (1989) is almost identical to that of the standard BFGS method – the only difference is in the matrix update. Instead of storing the matrices H_k, one stores a certain number, say m, of pairs $\{s_i, y_i\}$ that define them implicitly. The product $H_k g_k$ is obtained by performing a sequence of inner products involving g_k and the m most recent vector pairs $\{s_i, y_i\}$. After computing the new iterate, the oldest pair is deleted from the set $\{s_i, y_i\}$, and is replaced by the newest one. The algorithm therefore always keeps the m most recent pairs $\{s_i, y_i\}$ to define the iteration matrix. This approach is suitable for large problems because it has been observed in practice that small values of m (say $m \in [3, 7]$) give satisfactory results.

Let us describe the updating process in more detail. Suppose that the current iterate is x_k and that we have stored the m pairs $\{s_i, y_i\}$, $i = k-m, ..., k-1$. We choose a "basic matrix" $H_k^{(0)}$ (usually a diagonal matrix) and update it m times using the BFGS formula and the m pairs $\{s_i, y_i\}$, $i = k-m, ..., k-1$. From (1.2) we see that H_k can be written as

$$\begin{aligned} H_k &= \left(V_{k-1}^T \cdots V_{k-m}^T\right) H_k^{(0)} \left(V_{k-m} \cdots V_{k-1}\right) \\ &+ \rho_{k-m} \left(V_{k-1}^T \cdots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T \left(V_{k-m+1} \cdots V_{k-1}\right) \end{aligned}$$

$$
\begin{aligned}
&+ \quad \rho_{k-m+1}\left(V_{k-1}^T \cdots V_{k-m+2}^T\right) s_{k-m+1} s_{k-m+1}^T \left(V_{k-m+2} \cdots V_{k-1}\right) \\
&+ \quad \vdots \\
&+ \quad \rho_{k-1} s_{k-1} s_{k-1}^T .
\end{aligned} \tag{1.4}
$$

There is a recursive formula (Nocedal (1980)) that takes advantage of the symmetry of this expression to compute the product $H_k g_k$ efficiently. As a result, the computation of the search direction in the limited memory BFGS method for unconstrained optimization can be performed very economically.

It turns out, however, that in two respects this recursive formula is much less economical for some of the calculations required when constraints are present. First, when the constraints are sparse the recursion does not take good advantage of this sparsity. For example, if e_i is a unit vector, the computation of $H_k e_i$ is almost as expensive as the computation of $H_k g_k$. Second, many algorithms for constrained optimization require the direct Hessian approximation, $B_k = H_k^{-1}$ instead of the inverse BFGS approximation, H_k. However, there appears to be no analogous recursion for the Hessian approximation B_k and, as pointed out in Section 4.2, a straightforward implementation turns out to be quite costly.

After deriving our new quasi-Newton representations in Section 2, we show in Section 3 how they can be used in limited memory methods in a way that is efficient for unconstrained optimization, and gets around both of these difficulties in constrained optimization calculations.

Notation. The number of variables in the optimization problem is n, and the number of correction pairs used in the limited memory methods is m. The Hessian approximation is denoted by B_k, and the inverse Hessian approximation is H_k. The i-th unit vector is written as e_i. A diagonal matrix with diagonal elements $\theta_1, ..., \theta_n$ is denoted by $\mathrm{diag}[\theta_1, ..., \theta_n]$.

2. Compact Representations of BFGS Matrices

We will now describe new representations of the inverse and direct BFGS matrices, and show how to compute several types of matrix-vector products efficiently. In this section we will consider the updating process in a general setting, and will not restrict it to the case of limited memory methods.

Let us define the $n \times k$ matrices S_k and Y_k by

$$
S_k = [s_0, \ldots, s_{k-1}], \qquad Y_k = [y_0, \ldots, y_{k-1}]. \tag{2.1}
$$

We first prove a preliminary lemma on products of projection matrices that will be useful in subsequent analysis and is also interesting in its own right.

Lemma 2.1 *The product of a set of k projection matrices of the form (1.3) satisfies*

$$V_0 \cdots V_{k-1} = I - Y_k R_k^{-1} S_k^T, \tag{2.2}$$

where R_k is the $k \times k$ matrix

$$(R_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}. \tag{2.3}$$

Proof. Proceeding by induction we note that (2.2) holds for $k = 1$, because in this case the right hand side of (2.2) is

$$I - y_0 \frac{1}{s_0^T y_0} s_0^T = V_0. \tag{2.4}$$

Now we assume that (2.2) holds for some k, and consider $k+1$. If we write the matrix R_{k+1} as

$$R_{k+1} = \begin{bmatrix} R_k & S_k^T y_k \\ 0 & \frac{1}{\rho_k} \end{bmatrix},$$

we see that

$$R_{k+1}^{-1} = \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T y_k \\ 0 & \rho_k \end{bmatrix}. \tag{2.5}$$

This implies that

$$\begin{aligned} I - Y_{k+1} R_{k+1}^{-1} S_{k+1}^T &= I - \begin{bmatrix} Y_k & y_k \end{bmatrix} \begin{bmatrix} R_k^{-1} & -\rho_k R_k^{-1} S_k^T y_k \\ 0 & \rho_k \end{bmatrix} \begin{bmatrix} S_k^T \\ s_k^T \end{bmatrix} \\ &= I - Y_k R_k^{-1} S_k^T + \rho_k Y_k R_k^{-1} S_k^T y_k s_k^T - \rho_k y_k s_k^T \\ &= (I - Y_k R_k^{-1} S_k^T)(I - \rho_k y_k s_k^T). \end{aligned}$$

Using this with the inductive hypothesis of (2.2) we have that

$$\begin{aligned} V_0 \cdots V_k &= (I - Y_k R_k^{-1} S_k^T)(I - \rho_k y_k s_k^T) \\ &= (I - Y_{k+1} R_{k+1}^{-1} S_{k+1}^T), \end{aligned}$$

which establishes the product relation (2.2) for all k. □

It should be pointed out that this lemma holds for the product of any sequence of projections onto spaces of dimension $n-1$ and is a useful but little-known result. Essentially the same result is also mentioned by Walker (1988) in the context of products of Householder transformations. The lemma can be generalized to projections onto subspaces of arbitrary and different dimensions, in which case the matrix R_k becomes block upper triangular.

The following theorem gives a compact representation of the matrix H_k obtained after k BFGS updates. We will later see that this representation is often more convenient than (1.4).

Theorem 2.2 *Let H_0 be symmetric and positive definite and assume that the k pairs $\{s_i, y_i\}_{i=0}^{k-1}$ satisfy $s_i^T y_i > 0$. Let H_k be obtained by updating H_0 k times using the inverse BFGS formula (1.2) and the pairs $\{s_i, y_i\}_{i=0}^{k-1}$. Then*

$$H_k = H_0 + \left[\begin{array}{cc} S_k & H_0 Y_k \end{array} \right] \left[\begin{array}{cc} R_k^{-T}(D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{array} \right] \left[\begin{array}{c} S_k^T \\ Y_k^T H_0 \end{array} \right], \qquad (2.6)$$

where R_k is as given in (2.3) and D_k is the $k \times k$ diagonal matrix

$$D_k = \mathrm{diag}\left[s_0^T y_0, \ldots, s_{k-1}^T y_{k-1} \right]. \qquad (2.7)$$

Proof. We write the BFGS formula (1.2) as

$$H_k = M_k + N_k, \quad k \geq 1 \qquad (2.8)$$

where M_k and N_k are defined recursively by

$$\begin{cases} M_0 = H_0 \\ M_{k+1} = V_k^T M_k V_k, \end{cases} \qquad (2.9)$$

and

$$\begin{cases} N_1 = \rho_0 s_0 s_0^T \\ N_{k+1} = V_k^T N_k V_k + \rho_k s_k s_k^T. \end{cases} \qquad (2.10)$$

First note, from the definition of M_k and (2.2), that

$$\begin{aligned} M_k &= \left(V_{k-1}^T \cdots V_0^T \right) H_0 \left(V_0 \cdots V_{k-1} \right) \\ &= (I - S_k R_k^{-T} Y_k^T) H_0 (I - Y_k R_k^{-1} S_k^T). \end{aligned} \qquad (2.11)$$

Next, we will show by induction that

$$N_k = S_k R_k^{-T} D_k R_k^{-1} S_k^T. \qquad (2.12)$$

This is true for $k = 1$, for in this case the right hand side of (2.12) is $\rho_0 s_0 s_0^T$, which equals N_1. Now let us assume that (2.12) is true for k. Then, by the definition (2.10) of N,

$$N_{k+1} = V_k^T S_k R_k^{-T} D_k R_k^{-1} S_k^T V_k + \rho_k s_k s_k^T. \qquad (2.13)$$

To simplify this expression, we note from (1.3) and (2.5) that

$$\begin{aligned} R_k^{-1} S_k^T V_k &= R_k^{-1} S_k^T (I - \rho_k y_k s_k^T) \\ &= \left[\begin{array}{cc} R_k^{-1} & -\rho_k R_k^{-1} S_k^T y_k \end{array} \right] \left[\begin{array}{c} S_k^T \\ s_k^T \end{array} \right] \\ &= \left[\begin{array}{cc} R_k^{-1} & -\rho_k R_k^{-1} S_k^T y_k \end{array} \right] S_{k+1}^T \\ &= \left[\begin{array}{cc} I & 0 \end{array} \right] R_{k+1}^{-1} S_{k+1}^T. \end{aligned} \qquad (2.14)$$

Also, using (2.5) we can write s_k as

$$s_k = S_{k+1} R_{k+1}^{-T} e_{k+1} \frac{1}{\rho_k}. \tag{2.15}$$

Substituting this and (2.14) in (2.13), we have

$$\begin{aligned} N_{k+1} &= S_{k+1} R_{k+1}^{-T} \begin{bmatrix} I \\ 0 \end{bmatrix} D_k \begin{bmatrix} I & 0 \end{bmatrix} R_{k+1}^{-1} S_{k+1}^T + S_{k+1} R_{k+1}^{-T} \begin{bmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & & \frac{1}{\rho_k} \end{bmatrix} R_{k+1}^{-1} S_{k+1}^T \\ &= S_{k+1} R_{k+1}^{-T} \begin{bmatrix} D_k & 0 \\ 0 & \frac{1}{\rho_k} \end{bmatrix} R_{k+1}^{-1} S_{k+1}^T \\ &= S_{k+1} R_{k+1}^{-T} D_{k+1} R_{k+1}^{-1} S_{k+1}^T. \end{aligned}$$

This proves (2.12) for $k+1$.

Finally by expanding the expression

$$H_0 + \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \begin{bmatrix} R_k^{-T}(D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix}$$

we see that it is equal to $M_k + N_k$, where M_k and N_k are given by (2.11) and (2.12).

□

Note that the conditions $s_i^T y_i > 0 \;\; i = 0, ..., k-1$ ensure that R_k is nonsingular, so that (2.6) is well defined. Indeed it is well known (Fletcher (1987)) that the BFGS formula preserves positive definiteness if $s_i^T y_i > 0$ for all i.

Theorem 2.2 gives us a matrix representation of the inverse Hessian approximation H_k. We now present an analogous expression for the direct Hessian approximation B_k. The direct BFGS update formula, i.e. the inverse of (1.2) is given by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{2.16}$$

Theorem 2.3 *Let B_0 be symmetric and positive definite and assume that the k pairs $\{s_i, y_i\}_{i=0}^{k-1}$ satisfy $s_i^T y_i > 0$. Let B_k be obtained by updating B_0 k times using the direct BFGS formula (2.16) and the pairs $\{s_i, y_i\}_{i=0}^{k-1}$. Then*

$$B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}, \tag{2.17}$$

where L_k is the $k \times k$ matrix

$$(L_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \textit{if } i > j \\ 0 & \textit{otherwise} \end{cases}. \tag{2.18}$$

Proof. Let us write (2.6) as

$$H_k = H_0 + U_k C_k U_k^T, \tag{2.19}$$

where

$$U_k = \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix},$$

and

$$C_k = \begin{bmatrix} R_k^{-T}(D_k + Y_k^T H_0 Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{bmatrix}.$$

By direct multiplication we can verify that the inverse of C_k is

$$C_k^{-1} = \begin{bmatrix} 0 & -R_k \\ -R_k^T & -(D_k + Y_k^T H_0 Y_k) \end{bmatrix}. \tag{2.20}$$

Applying the Sherman-Morrison-Woodbury formula (Ortega and Rheinboldt (1970)) to (2.19) we obtain

$$\begin{aligned} B_k &= B_0 - B_0 U_k (I + C_k U_k^T B_0 U_k)^{-1} C_k U_k^T B_0 \\ &= B_0 - B_0 U_k (C_k^{-1} + U_k^T B_0 U_k)^{-1} U_k^T B_0. \end{aligned} \tag{2.21}$$

Now

$$\begin{aligned} U_k^T B_0 U_k &= \begin{bmatrix} S_k^T \\ Y_k^T H_0 \end{bmatrix} B_0 \begin{bmatrix} S_k & H_0 Y_k \end{bmatrix} \\ &= \begin{bmatrix} S_k^T B_0 S_k & S_k^T Y_k \\ Y_k^T S_k & Y_k^T H_0 Y_k \end{bmatrix}. \end{aligned}$$

Therefore using (2.20)

$$C_k^{-1} + U_k^T B_0 U_k = \begin{bmatrix} S_k^T B_0 S_k & S_k^T Y_k - R_k \\ Y_k^T S_k - R_k^T & -D_k \end{bmatrix}.$$

Note that the matrix L_k defined by (2.18) can be written as

$$L_k = S_k^T Y_k - R_k, \tag{2.22}$$

so that

$$C_k^{-1} + U_k^T B_0 U_k = \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}. \tag{2.23}$$

Substituting this into (2.21) we obtain (2.17).

□

In the next sections we will show that the new formulae (2.17) and (2.6), which at first appear rather cumbersome, are actually very convenient for some calculations arising in constrained optimization. Before doing so we make a remark concerning the implementation of (2.17).

The middle matrix in (2.17),

$$\begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix} \tag{2.24}$$

is indefinite. However we now show that its inversion can be carried out using the Cholesky factorization of a related matrix. First we re-order the blocks of (2.24) and note that

$$\begin{bmatrix} -D_k & L_k^T \\ L_k & S_k^T B_0 S_k \end{bmatrix} = \begin{bmatrix} D_k^{1/2} & 0 \\ -L_k D_k^{-1/2} & J_k \end{bmatrix} \begin{bmatrix} -D_k^{1/2} & D_k^{-1/2} L_k^T \\ 0 & J_k^T \end{bmatrix}, \tag{2.25}$$

where J_k is the lower triangular matrix that satisfies

$$J_k J_k^T = S_k^T B_0 S_k + L_k D_k^{-1} L_k^T. \tag{2.26}$$

The following result shows that J_k exists and is nonsingular.

Theorem 2.4 *If B_0 is positive definite and $s_i^T y_i > 0, i = 0, ..., k-1$, then the matrix $S_k^T B_0 S_k + L_k D_k^{-1} L_k^T$ is positive definite.*

Proof. From the definition (2.7) we see that D_k is positive definite and hence $S_k^T B_0 S_k + L_k D_k^{-1} L_k^T$ is positive semi-definite. Suppose that $u^T(S_k^T B_0 S_k + L_k D_k^{-1} L_k^T)u = 0$ for some vector u. Then $L_k^T u = 0$ and $S_k u = 0$, which in turn implies that $Y_k^T S_k u = 0$. Recalling (2.22) we have $Y_k^T S_k = L_k^T + R_k^T$, so that $R_k^T u = 0$. Since R_k^T is triangular with positive diagonal, we conclude that $u = 0$.

□

Therefore, only the Cholesky factorization of the $k \times k$ symmetric positive definite matrix $S_k^T B_0 S_k + L_k D_k^{-1} L_k^T$ needs to be computed, to implement (2.17). This is preferable to factorizing the indefinite $2k \times 2k$ matrix (2.24). We will discuss the implementation of (2.17) in more detail in section 3.2, in the context of limited memory methods.

3. Application to the Limited Memory Method.

Since we know that k BFGS updates can be written in the compact forms (2.6) and (2.17), it is easy to describe a limited memory implementation. We keep the m most recent correction pairs $\{s_i, y_i\}$ to implicitly define the iteration matrix. This set of pairs is refreshed at every iteration by removing the oldest pair and adding a newly generated pair. We assume that m is constant, but it is not difficult to adapt all the formulae of this section to the case when m changes at every iteration.

Suppose that at the current iterate x_k we wish to construct the inverse limited memory BFGS matrix H_k. We do so by implicitly updating $H_k^{(0)}$, the basic matrix, m times using the $2m$ vectors $\{s_{k-m}, \ldots, s_{k-1}\}$ and $\{y_{k-m}, \ldots, y_{k-1}\}$, which have been saved. Let us assume that $H_k^{(0)} = \gamma_k I$, for some positive scalar γ_k. From (2.6) we see that the resulting matrix is

$$H_k = \gamma_k I + \left[\begin{array}{cc} S_k & \gamma_k Y_k \end{array} \right] \left[\begin{array}{cc} R_k^{-T}(D_k + \gamma_k Y_k^T Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{array} \right] \left[\begin{array}{c} S_k^T \\ \gamma_k Y_k^T \end{array} \right], \tag{3.1}$$

where now

$$S_k = [s_{k-m}, \ldots, s_{k-1}], \qquad Y_k = [y_{k-m}, \ldots, y_{k-1}], \tag{3.2}$$

and where R_k and D_k are the $m \times m$ matrices

$$(R_k)_{i,j} = \begin{cases} (s_{k-m-1+i})^T (y_{k-m-1+j}) & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

and

$$D_k = diag\left[s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1} \right]. \tag{3.4}$$

After the new iterate x_{k+1} is generated, we obtain S_{k+1} by deleting s_{k-m} from S_k and adding the new displacement s_k. The matrix Y_{k+1} is updated in the same fashion.

This describes the general step when $k > m$. For the first few iterations, when $k \leq m$, we need only replace m by k in the formulae above. We have assumed that $H_k^{(0)} = \gamma_k I$ because this choice is common in practice (see Gilbert and Lemaréchal (1989) and Liu and Nocedal (1989)). Other formulae for the initial matrix could also be used, but would probably result in a more expensive computation.

A limited memory matrix based on the direct BFGS formula is also easily obtained. Let the basic matrix be of the form $B_k^{(0)} = \sigma_k I$, for some positive scalar σ_k. From (2.17) we see that if we update $B_k^{(0)}$ m times using the vectors $\{s_{k-m}, \ldots, s_{k-1}\}$ and $\{y_{k-m}, \ldots, y_{k-1}\}$, we obtain

$$B_k = \sigma_k I - \left[\begin{array}{cc} \sigma_k S_k & Y_k \end{array} \right] \left[\begin{array}{cc} \sigma_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{array} \right]^{-1} \left[\begin{array}{c} \sigma_k S_k^T \\ Y_k^T \end{array} \right], \tag{3.5}$$

where S_k, Y_k, D_k are given by (3.2) and (3.4), and where L_k is defined by

$$(L_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-1+j} & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}. \tag{3.6}$$

We now describe procedures for performing computations with these compact representations of limited memory BFGS matrices.

3.1. Computations involving H_k.

We consider several products involving the inverse limited memory matrix H_k. To save computations we will store, in addition to the two $n \times m$ matrices S_k and Y_k, the $m \times m$ matrices $Y_k^T Y_k$, R_k, and D_k. Since in practice m is very small, say $m \leq 7$, the storage space required by these three auxiliary matrices is negligible. In the operation counts given below we concentrate on multiplications since the arithmetic consists primarily of inner products, so that the number of additions is similar to the number of multiplications. We note that for the rest of this section S_k, Y_k, R_k, D_k, L_k are defined by (3.2)-(3.4) and (3.6).

Computation of $H_k g_k$.
This product defines the search direction in a limited memory method for unconstrained optimization. Since some of the calculations involved in the product $H_k g_k$ occur also in the update of H_k, it is efficient to consider both operations together.

At the k-th iteration of the limited memory algorithm for unconstrained optimization we must update our representation of H_{k-1} to get H_k, compute the search direction $-H_k g_k$ and perform a line search. To update H_{k-1} we delete a column from and add a new column to each of the matrices S_{k-1} and Y_{k-1}, and make corresponding updates to R_{k-1}, $Y_{k-1}^T Y_{k-1}$ and D_{k-1}. We will show that these updates can be done in $O(m^2)$ operations by storing a small amount of additional information. For example, from (3.3) we see that the new triangular matrix R_k is formed from R_{k-1} by deleting the first row and column, adding a new column on the right, which is given by

$$S_k^T y_{k-1} = S_k^T (g_k - g_{k-1}), \tag{3.7}$$

and adding a new row on the bottom, which is zero in its first $m-1$ components. It would appear that this requires mn multiplications. However, note from (3.1) that the vector $S_k^T g_k$ and the first $m-1$ components of $S_k^T g_{k-1}$ have to be calculated in the process of computing $H_k g_k$ and $H_{k-1} g_{k-1}$. Thus we may save the first $m-1$ components of $S_k^T g_{k-1}$ from the previous iteration, and we need only compute $s_{k-1}^T g_{k-1}$, which can be obtained with $O(m^2)$ work, as we will show below. Thus to compute $S_k^T y_{k-1}$ by the difference (3.7) will require only $O(m^2)$ operations. The matrix $Y_k^T Y_k$ can be updated in a similar way saving another mn multiplications.

An updating process that implements these savings in computation is as follows. At x_k, the following data has been saved from the previous iteration:

$$g_{k-1}^T g_{k-1},$$

$$s_i^T g_{k-1} \qquad i = k-m-1, \ldots, k-2, \qquad \text{(i.e. } S_{k-1}^T g_{k-1}\text{)}$$

and

$$y_i^T g_{k-1} \qquad i = k-m-1, \ldots, k-2 \qquad \text{(i.e. } Y_{k-1}^T g_{k-1}\text{)}.$$

Now we compute the quantities corresponding to the present iteration. We begin with

$$s_{k-1}^T g_{k-1} = -\lambda_{k-1} g_{k-1}^T H_{k-1} g_{k-1},$$

which by (3.1) is equal to

$$-\lambda_{k-1}\gamma_{k-1}g_{k-1}^T g_{k-1} - \lambda_{k-1} w_k^T \begin{bmatrix} R_{k-1}^{-T}(D_{k-1}+\gamma_{k-1}Y_{k-1}^T Y_{k-1})R_{k-1}^{-1} & -R_{k-1}^{-T} \\ -R_{k-1}^{-1} & 0 \end{bmatrix} w_k \quad (3.8)$$

where

$$w_k = \begin{bmatrix} S_{k-1}^T g_{k-1} \\ \gamma_{k-1} Y_{k-1}^T g_{k-1} \end{bmatrix}.$$

This requires only $O(m^2)$ operations since $g_{k-1}^T g_{k-1}$, $S_{k-1}^T g_{k-1}$ and $Y_{k-1}^T g_{k-1}$ have already been saved from the previous iteration.

Next we compute the inner products

$$g_k^T g_k,$$

$$s_i^T g_k \qquad i = k-m, \ldots, k-1, \qquad (\text{i.e. } S_k^T g_k)$$

and

$$y_i^T g_k \qquad i = k-m, \ldots, k-1, \qquad (\text{i.e. } Y_k^T g_k).$$

With this information, the new components of $R_k, Y_k^T Y_k$ and D_k, can be computed in $O(m)$ work by the formulae

$$s_i^T y_{k-1} = s_i^T g_k - s_i^T g_{k-1} \qquad i = k-m, \ldots, k-1, \quad (3.9)$$

$$y_i^T y_{k-1} = y_i^T g_k - y_i^T g_{k-1} \qquad i = k-m, \ldots, k-2, \quad (3.10)$$

$$y_{k-1}^T y_{k-1} = -g_k^T g_k + 2(g_k - g_{k-1})^T g_k + g_{k-1}^T g_{k-1}. \quad (3.11)$$

We now give a complete description of the procedure for updating H_k and computing $H_k g_k$.

Algorithm 3.1 (Step Computation for Unconstrained Minimization)

Let x_k be the current iterate. Given s_{k-1}, y_{k-1}, g_k, the matrices S_{k-1}, Y_{k-1}, R_{k-1}, $Y_{k-1}^T Y_{k-1}$, D_{k-1}, the vectors $S_{k-1}^T g_{k-1}$, $Y_{k-1}^T g_{k-1}$ and the scalar $g_{k-1}^T g_{k-1}$:

1. Update S_k, Y_k
2. Compute $g_k^T g_k, S_k^T g_k$, and $Y_k^T g_k$
3. Compute $s_{k-1}^T g_{k-1}$ by (3.8)
4. Update $R_k, Y_k^T Y_k$ and D_k with the aid of (3.9)-(3.11).
5. Compute γ_k; for example

$$\gamma_k = {y_{k-1}}^T s_{k-1} / {y_{k-1}}^T y_{k-1}. \quad (3.12)$$

6. Compute

$$p = \left[\begin{array}{c} R_k^{-T}(D_k + \gamma_k Y_k^T Y_k) R_k^{-1}(S_k^T g_k) - \gamma_k R_k^{-T}(Y_k^T g_k) \\ -R_k^{-1}(S_k^T g_k). \end{array} \right]$$

7. Compute

$$H_k g_k = \gamma_k g_k + \left[\begin{array}{cc} S_k & \gamma_k Y_k \end{array} \right] p.$$

In this procedure, step 2 requires $(2m+1)n$ multiplications; step 7 requires $(2m+1)n$ multiplications; step 5 depends on the formula used for γ_k (the choice (3.12) is free since both inner products have been stored); all other steps cost at most $O(m^2)$ multiplications, for a total of $(4m+2)n+O(m^2)$ multiplications. Note, however, that when this procedure is part of an algorithm using a line search procedure, the scalar $s_{k-1}^T g_{k-1}$ is also required for the line search, whereas $g_k^T g_k$ is likely to be needed to check the stopping conditions of the algorithm. Therefore the amount of extra work required to update H_k and compute the step direction is $4mn + O(m^2)$ in that case. Of course for large problems the term $4mn$ predominates.

As will be seen in Section 4.1 this is the same amount of work per iteration as required by the two-loop recursion described by Nocedal (1980), and as far as we know there is no more efficient way to implement the unconstrained limited memory BFGS method. Thus the two approaches are equally efficient for unconstrained problems, but, as pointed out in Section 4.1, the compact matrix representations derived in this paper are more economical when computing certain quantities arising in sparse constrained optimization calculations.

The product $H_k v$.
Let us consider the computation of the product $H_k v$, where v is an arbitrary vector. From (3.1) we see that this product is given by

$$H_k v = \gamma_k v + \left[\begin{array}{cc} S_k & \gamma_k Y_k \end{array} \right] \left[\begin{array}{cc} R_k^{-T}(D_k + \gamma_k Y_k^T Y_k) R_k^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{array} \right] \left[\begin{array}{c} S_k^T v \\ \gamma_k Y_k^T v \end{array} \right]. \tag{3.13}$$

To carry out the computation we first compute the products $S_k^T v$ and $Y_k^T v$, which together require $2mn$ multiplications. To multiply the resulting $2m$ vector by the middle $2m \times 2m$ matrix involves 3 solutions of triangular systems and one multiplication by an $m \times m$ matrix. Finally, it takes $2mn$ multiplications to multiply $[S_k \ \gamma_k Y_k]$ with the resulting $2m$ vector. Thus, if we include the product $\gamma_k v$ and ignore $O(m)$ operations, the whole computation requires $(4m+1)n + \frac{5}{2}m^2$ multiplications.

Products of the form $v^T H_k v$ and $u^T H_k v$.
Consider the weighted scalar product $v^T H_k v$ where v is an arbitrary vector, and where we assume that the vector $H_k v$ is not needed. Using (3.1) we have

$$v^T H_k v = \gamma_k v^T v + (R_k^{-1} S_k^T v)^T (D_k + \gamma_k Y_k^T Y_k)(R_k^{-1} S_k^T v) - 2\gamma_k v^T Y_k R_k^{-1} S_k^T v. \tag{3.14}$$

We first compute $S_k^T v$ and $Y_k^T v$, which requires $2mn$ multiplications. Next we solve a triangular system to get $R_k^{-1} S_k^T v$, which we save, multiply by the matrix $D_k + \gamma Y_k^T Y_k$, compute $v^T v$ and do some order m inner products. Thus the total cost of this computation is $(2m+1)n + \frac{3}{2}m^2 + O(m)$: roughly half of what the cost would be if we first computed $H_k v$ and then $v^T H_k v$.

If we wish to compute the product $u^T H_k v$ for two arbitrary vectors u and v the cost is more, since

$$\begin{aligned} u^T H_k v &= \gamma_k u^T v + (R_k^{-1} S_k^T u)^T (D_k + \gamma_k Y_k^T Y_k)(R_k^{-1} S_k^T v) - \gamma_k u^T Y_k R_k^{-1} S_k^T v \\ &\quad -\gamma_k u^T S_k R_k^{-T} Y_k^T v \end{aligned}$$

can be seen to require $(4m+1)n + 2m^2 + O(m)$ multiplications. This is only slightly less expensive than computing $H_k v$ and then taking the inner product of the result with u, which would cost $(4m+2)n + O(m^2)$ multiplications.

The Product $A^T H_k A$.

A related computation is the problem of computing the matrix $A^T H_k A$ where A is an $n \times t$ matrix with $t \leq n$. This computation occurs when solving the constrained nonlinear optimization problem,

$$\text{minimize} \quad f(x) \tag{3.15}$$

$$\text{subject to} \quad c(x) = 0 \tag{3.16}$$

with n variables and t constraints. This problem is frequently solved by the sequential quadratic programming method, which at every iteration solves a subproblem of the form

$$\text{minimize} \quad g_k^T d + \tfrac{1}{2} d^T B_k d \tag{3.17}$$

$$\text{subject to} \quad A_k^T d = -c_k, \tag{3.18}$$

where A_k is the matrix of constraint gradients at the current iterate x_k, c_k is a vector of length t, and $B_k = H_k^{-1}$ is an approximation to the Hessian of the Lagrangian of the problem. If A_k has full rank, the solution to (3.17)-(3.18) can be expressed as

$$d = -H_k(g_k + A_k \lambda) \tag{3.19}$$

where the Lagrange multiplier λ satisfies

$$(A_k^T H_k A_k)\lambda = -A_k^T H_k g_k + c_k. \tag{3.20}$$

Let us suppose that H_k is a limited memory matrix represented in the compact form (3.1). Then the matrix $A_k^T H_k A_k$ may be efficiently computed by first computing $S_k^T A_k$ and $Y_k^T A_k$, which require $2mnt$ multiplications, then $R_k^{-1} S_k^T A_k$, requiring $\frac{1}{2}m^2 t$ multiplications, and then computing

$$\gamma_k A_k^T A_k + (R_k^{-1} S_k^T A_k)^T (D_k + \gamma_k Y_k^T Y_k)(R_k^{-1} S_k^T A_k) - 2\gamma_k A_k^T Y_k R_k^{-1} S_k^T A_k, \tag{3.21}$$

which requires $m^2t+\frac{3}{2}mt^2+\frac{1}{2}(t^2+t)n+O((\max\{m,t\})^2)$ multiplications. Ignoring lower order terms, this is a total of

$$(2m+\tfrac{1}{2}t+\tfrac{1}{2})tn+\frac{3}{2}(m+t)mt$$

multiplications. As long as m and t are fairly small this is not extremely expensive and is much less than the cost of computing the matrix H_kA_k first, and then multiplying by A_k^T. To solve (3.20) requires the Cholesky factorization of $A_k^TH_kA_k$ which requires $\frac{1}{6}t^3$ multiplications. The other matrix vector products required in (3.19) and (3.20) cost about $(2t+4m)n$, if certain quantities computed in other parts of the procedure are saved and reused appropriately.

Operations with H_k and sparse constraints.
We now consider computations similar to those in the previous section but where the vectors and matrices multiplying H_k are sparse (but recall that H_k is dense). This is an important case because, even though g_k, S_k, and Y_k are not likely to be sparse, it is very common to have constrained optimization problems where the gradients of the constraints, and thus the matrix A in (3.18) are sparse. A special case in which we are very interested is the case of a minimization subject to bound constraints, where the matrices dealt with are actually submatrices of the identity. Significant reductions in computational cost result in such problems if efficient sparse storage is used.

The product H_ke_i requires $2mn+O(m^2)$ multiplications. This is easy to see from (3.13), since $S_k^Te_i$ and $Y_k^Te_i$ require only $O(m)$ indexing operations. For the same reason, we see from (3.14) that $e_i^TH_ke_i$ can be computed with $O(m^2)$ multiplications.

Consider now A^TH_kA in the case where A is an $n\times t$ sparse matrix with n_A non-zeros. We perform this computation by (3.21). The products S_k^TA and Y_k^TA together require $2mn_A$ multiplications. The back-solve $R_k^{-1}S_k^TA$ requires $\frac{1}{2}mt^2$ multiplications, and the rest of the operations require $2mt^2+m^2t+O((max\{m,t\})^2$ multiplications plus the operations of A^TA which cost at most tn_A multiplications. Thus the total is $O(max\{m,t\})n_A+(2t+\frac{3}{2}m)mt+O((max\{m,t\})^2)$. Thus we see that, while in the previous section the computational effort in most tasks was roughly proportional to the number of variables n, in the sparse case it is proportional to the number of non-zeros in the sparse array under consideration.

3.2. Operations with B_k

We now consider the direct Hessian approximation B_k. To take advantage of the decomposition (2.25), we rewrite (3.5) as

$$B_k=\sigma_kI-\begin{bmatrix} Y_k & \sigma_kS_k \end{bmatrix}\begin{bmatrix} -D_k^{1/2} & D_k^{-1/2}L_k^T \\ 0 & J_k^T \end{bmatrix}^{-1}\begin{bmatrix} D_k^{1/2} & 0 \\ -L_kD_k^{-1/2} & J_k \end{bmatrix}^{-1}\begin{bmatrix} Y_k^T \\ \sigma_kS_k^T \end{bmatrix}, \tag{3.22}$$

where J_k is defined by (2.26). We use this expression, both in the sparse and dense case, to compute several products involving B_k.

Update of B_k and the product $B_k v$.
This computation is required when applying limited memory methods to solve constrained optimization problem. It occurs, for example, in the algorithm for nonlinearly constrained problems developed by Mahidhara and Lasdon (1990), and in the primal limited memory algorithm for bound constrained optimization described by Byrd, Lu and Nocedal (1993).

The following procedure, which is based on the representation (3.22), describes in detail the k-th step of an iteration that first updates B_k and then computes the product $B_k v$ for an arbitrary vector v.

Algorithm 3.2

Let x_k be the current iterate, and assume that the matrices S_{k-1}, Y_{k-1}, L_{k-1}, $S_{k-1}^T S_{k-1}$, and D_{k-1} have been stored. The vectors s_{k-1}, y_{k-1} have just been computed, and the vector v is given.

1. Obtain S_k, Y_k, by updating S_{k-1} and Y_{k-1}.
2. Compute $L_k, S_k^T S_k$ and D_k.
3. Compute σ_k; for example
$$\sigma_k = {y_{k-1}}^T s_{k-1} / {s_{k-1}}^T s_{k-1}. \tag{3.23}$$
4. Compute the Cholesky factorization of $\sigma_k S_k^T S_k + L_k D_k^{-1} L_k^T$ to obtain $J_k J_k^T$.
5. Compute
$$p = \begin{bmatrix} Y_k^T v \\ \sigma_k S_k^T v \end{bmatrix}.$$
6. Perform a forward and then a backward solve to obtain
$$p := \begin{bmatrix} -D_k^{1/2} & D_k^{-1/2} L_k^T \\ 0 & J_k^T \end{bmatrix}^{-1} \begin{bmatrix} D_k^{1/2} & 0 \\ -L_k D_k^{-1/2} & J_k \end{bmatrix}^{-1} p.$$
7. Compute
$$B_k v = \sigma_k v - \begin{bmatrix} Y_k & \sigma_k S_k^T \end{bmatrix} p.$$

The first step of this procedure, in which the oldest columns of the matrices S_{k-1}, Y_{k-1} are replaced by the vectors s_{k-1}, and y_{k-1}, does not require any arithmetic. Step 2 requires $2m$ inner products to form the new columns of matrices L_k, $S_k^T S_k$ and D_k, which cost $2mn$ multiplications. The choice of σ_k in step 3 costs only one multiplication since both ${y_{k-1}}^T s_{k-1}$ and ${s_{k-1}}^T s_{k-1}$ have been calculated in step 2. In step 4 the

Cholesky factorization of the positive definite matrix $\sigma_k S_k^T S_k + L_k D_k^{-1} L_k^T$ costs $O(m^3)$ multiplications. Step 5 costs $2mn$ multiplications. The forward and the backward solves of $2m \times 2m$ triangular systems in step 6 cost $O(m^2)$ multiplications. Step 7 costs $(2m+1)n$ multiplications. In summary, this procedure costs $2mn + O(m^3)$ multiplications from step 1 to step 4, where the matrix B_k is defined; and costs $(4m+1)n + O(m^2)$ multiplications from step 5 to step 7, where the product $B_k v$ is calculated.

The weighted scalar product $v^T B_k v$.
This product occurs, for example, in the conjugate gradient inner-iteration as well as in the Cauchy point computation of the primal algorithm described by Byrd, Lu and Nocedal (1993). Using (3.22) we have

$$v^T B_k v = \sigma_k v^T v - v^T W_k^T \begin{bmatrix} -D_k^{1/2} & D_k^{-1/2} L_k^T \\ 0 & J_k^T \end{bmatrix}^{-1} \begin{bmatrix} D_k^{1/2} & 0 \\ -L_k D_k^{-1/2} & J_k \end{bmatrix}^{-1} W_k v, \qquad (3.24)$$

where

$$W_k = \begin{bmatrix} Y_k^T \\ \sigma_k S_k^T \end{bmatrix}.$$

We first compute and store the matrix vector products $Y_k^T v, \sigma_k S_k^T v$, which determine $W_k v$, and which require $2mn$ multiplications. Then we solve two $2m \times 2m$ triangular systems, and compute the scalar product of two $2m$-vectors; all of these cost at most $O(m^2)$ multiplications. The last part is to compute $\sigma_k v^T v$, and subtract the previously computed scalar from it. The total cost of this computation is $(2m+1)n + O(m^2)$ multiplications. Of course in the case $v = g_k$, which is often required, using previously computed quantities form the computation of H_k would allow this to be reduced to $O(m^2)$.

Sparse computations with B_k
Calculations involving the product of B_k and sparse vectors involve savings similar to those involving H_k; for example, computing $B_k e_i$ requires $2mn + O(m^3)$ multiplications. A special but important sparse case concerns minimization problems subject to bound constraints, in which the constraint gradients are submatrices of the identity matrix. Minimizing over a subspace in that case involves computations with the reduced Hessian approximation $\hat{B}_k = Z^T B_k Z$, where Z is an $n \times \hat{t}$ matrix whose columns are unit vectors. Thus the subspace problem is of size $\hat{t}$.

To express $\hat{B}_k$ we use (3.22) to obtain

$$\hat{B}_k = \sigma_k \hat{I} - \begin{bmatrix} \hat{Y}_k & \sigma_k \hat{S}_k \end{bmatrix} \begin{bmatrix} -D_k^{1/2} & D_k^{-1/2} L_k^T \\ 0 & J_k^T \end{bmatrix}^{-1} \begin{bmatrix} D_k^{1/2} & 0 \\ -L_k D_k^{-1/2} & J_k \end{bmatrix}^{-1} \begin{bmatrix} \hat{Y}_k^T \\ \sigma_k \hat{S}_k^T \end{bmatrix},$$

where $\hat{I} = Z^T Z$ is the identity matrix of size $\hat{t}$, and $\hat{Y}_k = Z^T Y_k$ and $\hat{S}_k = Z^T S_k$ are $\hat{t} \times m$ submatrices of Y_k and S_k. The procedure of multiplying the reduced Hessian $\hat{B}_k$ by an arbitrary $\hat{t}$-vector $\hat{v}$ is similar to steps 5 to 7 of Algorithm 3.2 and costs $(4m+1)\hat{t} + O(m^2)$

multiplications. Similarly, the weighted scalar product $\hat{v}^T \hat{B}_k \hat{v}$ costs $(2m+1)\hat{t} + O(m^2)$ multiplications.

In this case we see significant reductions in computational cost, resulting in work proportional to $\hat{t}$ rather than to n.

4. Alternative Formulae.

For the sake of completeness we now review two other known approaches for handling limited memory matrices. The first approach exploits the symmetry and structure of (1.4), giving rise to an efficient two-loop recursion for computing products using the inverse Hessian approximation. The second approach is for the direct BFGS update and consists of a straightforward sequence of multiplications.

4.1. The Two-Loop Recursion

The following recursive formula computes the step direction $H_k g_k$ for unconstrained minimization. It is given in Nocedal (1980) and is based on the recursion developed by Matthies and Strang (1979) for the standard BFGS method. As before, H_k represents a limited memory BFGS approximation of the inverse Hessian. It is obtained by applying m updates to a basic matrix $H_k^{(0)}$ using the m most recent correction pairs, which we label for simplicity $(s_0, y_0), ..., (s_{m-1}, y_{m-1})$.

1. $q = g_k$

2. For $i = m-1, \ldots, 0$
$$\left[\begin{array}{lcll} \alpha_i & = & \rho_i s_i^T q & \quad (\text{store } \alpha_i) \\ q & := & q - \alpha_i y_i & \end{array}\right.$$

3. $r = H_k^{(0)} q$

4. For $i = 0, 1, \ldots, m-1$
$$\left[\begin{array}{lcl} \beta & = & \rho_i y_i^T r \\ r & := & r + s_i(\alpha_i - \beta_i) \end{array}\right.$$

5. $H_k g_k = r$

Excluding step 3, this algorithm requires $4mn$ multiplications; if $H_k^{(0)}$ is diagonal then n additional multiplications are needed. When used for unconstrained minimization the computation and storage cost is thus essentially the same as using formula (2.6) implemented as described in Section 3.1, as long as $H_k^{(0)}$ is a scalar multiple of the identity. However, the two loop recursion has the advantage that the multiplication by the basic matrix $H_k^{(0)}$ is isolated form the rest of the computations. As a result the two-loop recursion will be less expensive than (2.6) in the case when $H_k^{(0)}$ differs from $H_{k-1}^{(0)}$

by more than a simple scalar multiplication, since the entire matrix $Y_k^T H_0^{(k)} Y_k$ would then have to be updated.

However, the two-loop recursion cannot be efficiently adapted for sparse projections. Let us consider for example the product $H_k e_i$, which can be obtained by replacing g_k with e_i in the two-loop recursion. Since the vectors s_i and y_i are in general not sparse, we see that only the computation of α_{m-1} in step 2 results in savings. Thus steps 2 and 4 require $(4m-1)n$ multiplications – almost the same as in the dense case.

We should also mention that while the compact form (2.6) has an analog (2.17) for the direct update, we know of no procedure analogous to the two loop recursion that can compute the direct update from $B_k^{(0)}$, S_k, and Y_k in $O(mn)$ operations.

Mathematically, the relation of the two-loop recursion to (2.6) can be seen if we note that (2.6) can be expressed

$$H_k = (I - S_k R_k^{-T} Y^T) H_k^{(0)} (I - Y_k R_k^{-1} S_k^T) + S_k R_k^{-T} D_k R_k^{-1} S_k^T.$$

The vector made up of the coefficients α_i can then be seen to be $R_k^{-1} S_k^T g_k$, and the final value of the vector q is $(I - Y_k R_k^{-1} S_k^T) g_k$. Note that in the two-loop procedure everything is computed afresh at each iteration, thus making it easier to change parameters such as $H_k^{(0)}$, while implementing (2.6) involves saving and updating more computed quantities, thus making information such as sparse projections of H more immediately accessible.

A close examination of the two-loop recursion reveals that it is similar in structure to computations of gradients by means of the adjoint method (or the reverse mode of automatic differentiation (Griewank (1989)). In fact Gilbert and Nocedal (1991) show that there is a precise relationship between these two algorithms: the two-loop recursion can be obtained by applying the adjoint method to compute the gradient of the function $h(g) = \frac{1}{2} g^T H_k g$ with respect to its argument g, where H_k is the limited memory BFGS matrix. The scalars α_i, which are saved during the first loop, correspond to the quantities referred to as the adjoint variables in the optimal control literature.

4.2. A Straightforward Approach.

The direct BFGS formula (2.16) can be written as

$$B_{k+1} = B_k - a_k a_k^T + b_k b_k^T, \tag{4.1}$$

where

$$a_k = \frac{B_k s_k}{(s_k^T B_k s_k^T)^{\frac{1}{2}}}, \qquad b_k = \frac{y_k}{(y_k^T s_k)^{\frac{1}{2}}}.$$

A straightforward implementation of the limited memory method consists of saving these intermediate vectors a_i and b_i to define the iteration matrix. It has been used by several authors including Mahidhara and Lasdon (1990).

In a typical iteration k, the matrix B_k is obtained by updating a starting matrix $B_k^{(0)}$ m times using the m most recent pairs, which we denote for simplicity,

$$(s_0, y_0), \ldots, (s_{m-1}, y_{m-1}).$$

From (4.1) we see that B_k can be written as

$$B_k = B_k^{(0)} + \sum_{i=0}^{m-1} [b_i b_i^T - a_i a_i^T], \tag{4.2}$$

where the vectors a_i, b_i can be computed by means of the following formula:

For $k = 0, 1, \ldots, m-1$

1.
$$b_k = y_k / (y_k^T s_k)^{\frac{1}{2}} \tag{4.3}$$

2.
$$a_k = B_k^{(0)} s_k + \sum_{i=0}^{k-1} \left[(b_i^T s_k) b_i - (a_i^T s_k) a_i \right] \tag{4.4}$$

3.
$$a_k := a_k / (s_k^T a_k)^{\frac{1}{2}}. \tag{4.5}$$

At the next iteration we repeat this process, except that the pair (s_0, y_0) is replaced by the new pair (s_m, y_m). The vectors a_i need to be recomputed form scratch since they all depend on the deleted pair (s_0, y_0). On the other hand, the vectors b_i and the inner products $b_i^T s_k$ can be saved from the previous iteration, and only the new values b_m and $b_i^T s_m$ need to be computed. Taking this into account, and assuming that $B_k^{(0)} = I$ we find that approximately

$$3/2\; m^2 n \text{ multiplications,}$$

are needed to determine the limited memory matrix.

To compute $B_m v$, for some vector $v \in R^n$, using (4.2) requires $4mn$ multiplications. This approach is therefore less efficient than that based on the compact matrix representation described in section 3.2. Indeed, whereas the product $B_k v$ costs the same in both cases, updating the representation of the limited memory matrix using the compact form requires only $2mn$ multiplications, compared to $3/2\; m^2 n$ multiplications needed by the approach described in this section.

5. Compact Representation of SR1 Matrices.

In this section we develop compact representations of matrices generated by the symmetric rank-one (SR1) formula. These representations are similar to the ones derived for the BFGS formula, but under some conditions require less storage.

The SR1 update formula is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}; \tag{5.1}$$

see for example Fletcher (1987). Note that this update is well defined only if the denominator $(B_k s_k - y_k)^T s_k$ is nonzero. In recent implementations of the SR1 method, the update is simply skipped if this denominator is very small relative to $\|s_k\|\|B_k s_k - y_k\|$ (Conn, Gould and Toint (1988), Khalfan, Byrd and Schnabel (1993)). Since the SR1 update does not have the property of hereditary positive definiteness, there is no reason to enforce the curvature condition $s_k^T y_k > 0$ as with BFGS updating, and we will thus consider a sequence of updates to an arbitrary matrix B_0 subject only to the assumption that the update is well defined.

Theorem 5.1 *Let the symmetric matrix B_0 be updated k times by means of the SR1 formula (5.1) using the pairs $\{s_i, y_i\}_{i=0}^{k-1}$, and assume that each update is well defined, i.e. $s_j^T(B_j s_j - y_j) \neq 0; j = 0, ..., k-1$. Then the resulting matrix B_k is given by*

$$B_k = B_0 + (Y_k - B_0 S_k)(D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1}(Y_k - B_0 S_k)^T, \tag{5.2}$$

where S_k, Y_k, D_k, and L_k are as defined in (2.1), (2.7) and (2.18), and the matrix $M_k \equiv (D_k + L_k + L_k^T - S_k^T B_0 S_k)$ is nonsingular.

Proof. We proceed by induction. When $k = 1$ the right hand side of (5.2) is

$$B_0 + (y_0 - B_0 s_0)\frac{1}{(y_0 - B_0 s_0)^T s_0}(y_0 - B_0 s_0)^T = B_1.$$

Let us now assume that (5.2) holds for some k. Define

$$Q_k = [q_0, \ldots, q_{k-1}] = Y_k - B_0 S_k, \tag{5.3}$$

and

$$M_k = D_k + L_k + L_k^T - S_k^T B_0 S_k. \tag{5.4}$$

Therefore

$$B_k = B_0 + Q_k M_k^{-1} Q_k^T.$$

Applying the SR1 update (5.1) to B_k we have

$$\begin{aligned}
B_{k+1} &= B_0 + Q_k M_k^{-1} Q_k^T + \frac{(y_k - B_0 s_k - Q_k M_k^{-1} Q_k^T s_k)(y_k - B_0 s_k - Q_k M_k^{-1} Q_k^T s_k)^T}{(y_k - B_0 s_k)^T s_k - s_k^T Q_k M_k^{-1} Q_k^T s_k} \\
&= B_0 + Q_k M_k^{-1} Q_k^T + \frac{(q_k - Q_k M_k^{-1} w_k)(q_k - Q_k M_k^{-1} w_k)^T}{q_k^T s_k - w_k^T M_k^{-1} w_k} \\
&= B_0 + \Big[q_k q_k^T - q_k (w_k^T M_k^{-1}) Q_k^T - Q_k (M_k^{-1} w_k) q_k^T \\
&\quad + Q_k (\delta_k M_k^{-1} + M_k^{-1} w_k w_k^T M_k^{-1}) Q_k^T \Big] / \delta_k,
\end{aligned}$$

where we have defined

$$w_k = Q_k^T s_k, \tag{5.5}$$

and where the denominator

$$\begin{aligned} \delta_k &\equiv q_k^T s_k - w_k^T M_k^{-1} w_k \\ &= (y_k - B_k s_k)^T s_k \end{aligned} \tag{5.6}$$

is non-zero by assumption. We may express this as

$$B_{k+1} = B_0 + \frac{1}{\delta_k} [Q_k \; q_k] \left[\begin{array}{cc} M_k^{-1}(\delta_k I + w_k w_k^T M_k^{-1}) & -M_k^{-1} w_k \\ -w_k^T M_k^{-1} & 1 \end{array} \right] \left[\begin{array}{c} Q_k^T \\ \\ q_k^T \end{array} \right]. \tag{5.7}$$

Now, from definitions (5.3), (5.4) and (5.5) we see that the new matrix M_{k+1} is given by

$$M_{k+1} = \left[\begin{array}{cc} M_k & w_k \\ w_k^T & q_k^T s_k \end{array} \right],$$

and by direct multiplication, using (5.3), (5.5) and (5.6), we see that

$$\left[\begin{array}{cc} M_k & w_k \\ w_k^T & q_k^T s_k \end{array} \right] \left[\begin{array}{cc} M_k^{-1}(\delta_k I + w_k w_k^T M_k^{-1}) & -M_k^{-1} w_k \\ -w_k^T M_k^{-1} & 1 \end{array} \right] \frac{1}{\delta_k} = I. \tag{5.8}$$

Therefore M_{k+1} is invertible, with M_{k+1}^{-1} given by the second matrix in (5.8), but this is the matrix appearing in (5.7). Thus, we see that (5.7) is equivalent to equation (5.2) with k replaced by $k+1$, which observation establishes the result.

□

Since the SR1 method is self dual, the inverse formula can be obtained simply by replacing B, s, y by H, y, s respectively (see Dennis and Schnabel (1983)). Alternatively, if B_k is invertible, application of the Sherman-Morrison-Woodbury formula to (5.2) shows the inverse of B_k is given by

$$H_k = H_0 + (S_k - H_0 Y_k)(R_k + R_k^T - D_k - Y_k^T H_0 Y_k)^{-1}(S_k - H_0 Y_k)^T. \tag{5.9}$$

However, in the context of unconstrained optimization, since the SR1 update is not always positive definite this formula is not as likely to be useful in step computation as is the inverse BFGS update.

It should be clear how to develop limited memory SR1 methods. In (5.2) we replace B_0 with the basic matrix at the k-th iteration, which we denoted earlier by $B_k^{(0)}$, and S_k and Y_k should now contain the m most recent corrections, as in (3.2). Savings in storage can be achieved if $B_k^{(0)}$ is kept fixed for all k, for in this case the only n-vectors one needs to store are the m columns of Q_k. This would result also in some savings in the cost of updating the matrix M_k, depending on the step computation strategy used. On the other hand, if $B_k^{(0)}$ is a scalar multiple of the identity and, as is often the case, one wants to change the scalar at each iteration, then both S_k and Y_k must be stored separately, and the storage and updating costs of the limited memory SR1 and BFGS methods are similar.

We will not give detailed algorithms for computing products involving limited memory SR1 matrices because the ideas are very similar to those described in the previous section. One point, however, that is worth discussing is how to compute the denominator in (5.1), at each stage of the limited memory updating, to determine if the update should be skipped. The condition

$$s_j^T(B_j s_j - y_j) \neq 0, \quad j = 0, ..., k-1, \tag{5.10}$$

can be expensive to test. Note however that (5.10) is equivalent to the nonsingularity of the principal minors of M_k. Thus, when using the form (5.2) in a limited memory method, the condition (5.10) could be tested when computing a triangular factorization of M_k without pivoting, with the test for a zero on the diagonal of the factor being made relative to the magnitude of Q_k and S_k. Skipping an update would correspond to deleting the corresponding row and column of M_k.

6. Representation of Broyden Matrices for Nonlinear Equations.

A widely used secant approximation to the Jacobian matrix of a system of nonlinear equations,

$$F(x) = 0, \quad F : \Re^n \to \Re^n, \tag{6.1}$$

is the Broyden update (Broyden (1965)),

$$A_{k+1} = A_k + \frac{(y_k - A_k s_k)s_k^T}{s_k^T s_k}. \tag{6.2}$$

Here $s_k = x_{k+1} - x_k$, $y_k = F(x_{k+1}) - F(x_k)$, and A_k is the approximation to the Jacobian of F. In this section we describe compact expressions of Broyden matrices that are similar to those given for BFGS and SR1. As before, we define

$$S_k = [s_0, \ldots, s_{k-1}], \qquad Y_k = [y_0, \ldots, y_{k-1}], \tag{6.3}$$

and we assume that the vectors s_i are non-zero.

Theorem 6.1 *Let A_0 be a nonsingular starting matrix, and let A_k be obtained by updating A_0 k times using Broyden's formula (6.2) and the pairs $\{s_i, y_i\}_{i=0}^{k-1}$. Then*

$$A_k = A_0 + (Y_k - A_0 S_k)N_k^{-1}S_k^T, \tag{6.4}$$

where N_k is the $k \times k$ matrix

$$(N_k)_{i,j} = \begin{cases} s_{i-1}^T s_{j-1} & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}. \tag{6.5}$$

Proof. It is easy to show (using induction) that A_k can be written as

$$A_k = B_k + C_k, \tag{6.6}$$

where B_k and C_k are defined recursively by

$$\begin{cases} B_0 = A_0 \\ B_{k+1} = B_k(I - \rho_k s_k s_k^T) \quad \forall k \geq 0, \end{cases} \tag{6.7}$$

and

$$\begin{cases} C_0 = 0 \\ C_{k+1} = C_k(I - \rho_k s_k s_k^T) + \rho_k y_k s_k^T \quad \forall k \geq 0, \end{cases} \tag{6.8}$$

and where

$$\rho_k = 1/s_k^T s_k.$$

Considering first B_k we note that it can be expressed as the product of B_0 with a sequence of projection matrices,

$$B_k = B_0(I - \rho_0 s_0 s_0^T) \cdots (I - \rho_{k-1} s_{k-1} s_{k-1}^T). \tag{6.9}$$

Now we apply Lemma 2.1, with $y := s$ in the definition (1.3), to this product of projections to yield the relation

$$B_k = A_0 - A_0 S_k N_k^{-1} S_k^T, \tag{6.10}$$

for all $k \geq 1$.

Next we show by induction that C_k has the compact representation

$$C_k = Y_k N_k^{-1} S_k^T. \tag{6.11}$$

By the definition (6.8), we have that $C_1 = y_0 \rho_0 s_0^T$, which agrees with (6.11) for $k = 1$. Assume now that (6.11) holds for k. Then by (6.8),

$$\begin{aligned} C_{k+1} &= Y_k N_k^{-1} S_k^T (I - \rho_k s_k s_k^T) + \rho_k y_k s_k^T \\ &= Y_k N_k^{-1} S_k^T - \rho_k Y_k N_k^{-1} S_k^T s_k s_k^T + \rho_k y_k s_k^T \\ &= \begin{bmatrix} Y_k & y_k \end{bmatrix} \begin{bmatrix} N_k^{-1} & -\rho_k N_k^{-1} S_k^T s_k \\ 0 & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ s_k^T \end{bmatrix} + \begin{bmatrix} Y_k & y_k \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & \rho_k \end{bmatrix} \begin{bmatrix} S_k^T \\ s_k^T \end{bmatrix} \\ &= Y_{k+1} \begin{bmatrix} N_k^{-1} & -\rho_k N_k^{-1} S_k^T s_k \\ 0 & \rho_k \end{bmatrix} S_{k+1}. \end{aligned} \tag{6.12}$$

Note, however, that

$$\begin{bmatrix} N_k^{-1} & -\rho_k N_k^{-1} S_k^T s_k \\ 0 & \rho_k \end{bmatrix} \begin{bmatrix} N_k & S_k^T s_k \\ 0 & \frac{1}{\rho_k} \end{bmatrix} = I, \tag{6.13}$$

which implies that the second matrix on the right hand side of (6.12) is N_{k+1}^{-1}. By induction this establishes (6.11). Finally, substituting (6.10) and (6.11) in (6.6), we obtain (6.4).

□

We now derive a compact representation of the inverse Broyden update which is given by

$$A_{k+1}^{-1} = A_k^{-1} + \frac{(s_k - A_k^{-1} y_k) s_k^T A_k^{-1}}{s_k^T A_k^{-1} y_k} \tag{6.14}$$

(see for example Dennis and Schnabel (1983)).

Theorem 6.2 *Let A_0^{-1} be a nonsingular starting matrix, and let A_k^{-1} be obtained by updating A_0^{-1} k times using the inverse Broyden formula (6.14) and the pairs $\{s_i, y_i\}_{i=0}^{k-1}$. Then*

$$A_k^{-1} = A_0^{-1} - (A_0^{-1} Y_k - S_k)(M_k + S_k^T A_0^{-1} Y_k)^{-1} S_k^T A_0^{-1}, \tag{6.15}$$

where S_k and Y_k are given by (6.3) and M_k is the $k \times k$ matrix

$$(M_k)_{i,j} = \begin{cases} -s_{i-1}^T s_{j-1} & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}. \tag{6.16}$$

Proof. Let

$$U = Y_k - A_0 S_k, \qquad V^T = N_k^{-1} S_k^T,$$

so that (6.4) becomes

$$A_k = A_0 + U V^T.$$

Applying the Sherman-Morrison-Woodbury formula we obtain

$$\begin{aligned} A_k^{-1} &= A_0^{-1} - A_0^{-1} U (I + V^T A_0^{-1} U)^{-1} V^T A_0^{-1} \\ &= A_0^{-1} - A_0^{-1} (Y_k - A_0 S_k)(I + N_k^{-1} S_k^T A_0^{-1} (Y_k - A_0 S_k))^{-1} N_k^{-1} S_k^T A_0^{-1} \\ &= A_0^{-1} - (A_0^{-1} Y_k - S_k)(N_k + S_k^T A_0^{-1} Y_k - S_k^T S_k)^{-1} S_k^T A_0^{-1}. \end{aligned}$$

By (6.5) and (6.16) we have that $N_k - S_k^T S_k = M_k$, which gives (6.15).

□

Note that since we have assumed that all the updates given by (6.14) exist, we have implicitly assumed the nonsingularity of A_k. This nonsingularity along with the Sherman-Morrison formula ensures that $(M_k + S_k^T A_0^{-1} Y_k)$ is nonsingular.

These representations of Broyden matrices have been used by Biegler, Nocedal and Schmid (1993) to approximate a portion of the Hessian of the Lagrangian in a successive quadratic programming method for constrained optimization.

7. Relation to Multiple Secant Updates.

There is a close algebraic correspondence, and in certain special cases an equivalence, between the representations of a sequence of quasi-Newton updates that have been discussed in this paper, and multiple secant updates that have previously been discussed by several authors including Barnes (1965), Gay and Schnabel (1978), Schnabel (1983), and Khalfan (1989). In this section we briefly discuss this correspondence, for the BFGS, SR1, and Broyden updates. We also make a few comments about the tradeoffs between using these two types of updates. In additional to the notation of the preceding sections, we use the notation that $\bar{R}_k$ is the $k \times k$ matrix that is the strict upper triangle of $S_k^T Y_k$, i.e. $\bar{R}_k = R_k - D_k$ where R_k and D_k are defined by (2.3) and (2.7). Thus

$$S_k^T Y_k = L_k + D_k + \bar{R}_k \tag{7.1}$$

where L_k is defined in (2.18).

Multiple secant updates are updates that enforce the last k secant equations, i.e. in the notation of Section 1 $B_k S_k = Y_k$ or $H_k Y_k = S_k$. While the papers mentioned above generally consider using multiple secant update to update B_k to B_{k+1}, analogous updates to those considered in this paper would arise from using multiple secant updates to update B_0 to B_k or H_0 to H_k. This is the context in which we consider multiple secant updates in this section.

In this context, the multiple secant version of the direct BFGS update applied to B_0 is given by

$$B_k = B_0 + Y_k (Y_k^T S_k)^{-1} Y_k^T - B_0 S_k (S_k^T B_0 S_k)^{-1} S_k^T B_0 \tag{7.2}$$

or using a representation analogous to (2.17),

$$B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & 0 \\ 0 & -Y_k^T S_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}, \tag{7.3}$$

(assuming $k \le n$). The matrix B_k given by (7.2) always obeys the k secant equations $B_k S_k = Y_k$. Schnabel (1983) shows that, assuming B_0 is symmetric and positive definite, B_k is symmetric if and only if $Y_k^T S_k$ is symmetric, and in addition B_k is positive definite if and only if $Y_k^T S_k$ is positive definite. These conditions are satisfied if $f(x)$ is a positive definite quadratic, but not in general otherwise. Schnabel (1983) discusses ways to perturb Y_k to $\tilde{Y}_k$ so that $\tilde{Y}_k^T S_k$ is symmetric and positive definite, at the cost of no longer exactly satisfying the original secant equations other than the most recent. These perturbations have some relation to the comparisons of this section, and we will return to them shortly.

By comparing the multiple secant update (7.3) and the representation for k consecutive, standard BFGS updates (2.17), it is clear that these two formulas are very similar algebraically. It is also immediate that if $Y_k^T S_k = D_k$, the multiple BFGS update to B_0 is equivalent to performing k standard BFGS updates. This condition, which means that $s_i^T y_j = 0$ for all $i \neq j$, is satisfied if $f(x)$ is quadratic and the step directions are mutually conjugate, but not in general otherwise. In general, the two formulas (2.17) and (7.3) result in different matrices B_k.

Identical comments are true regarding the BFGS update to the inverse Hessian. The inverse form of the multiple BFGS update (7.3) is

$$H_k = H_0 + \left[\begin{array}{cc} S_k & H_0 Y_k \end{array} \right] \left[\begin{array}{cc} W_k^{-T} + W_k^{-1}(Y_k^T H_0 Y_k) W_k^{-T} & -W_k^{-1} \\ -W_k^{-T} & 0 \end{array} \right] \left[\begin{array}{c} S_k^T \\ Y_k^T H_0 \end{array} \right] \qquad (7.4)$$

where $W_k = Y_k^T S_k$. Again, assuming H_0 is positive definite, this matrix is symmetric and positive definite if and only if $Y_k^T S_k$ is symmetric and positive definite. Again, the algebraic forms for (7.4) and (2.6) are very similar, and by comparing these equations and recalling definitions (2.3) and (2.7), it is immediate that the updates are identical if $Y_k^T S_k = D_k$, and in general are different otherwise.

From these comparisons, one can see that in the context of limited memory methods, the multiple BFGS updates (7.3) or (7.4) would offer similar algebraic efficiencies to the representations (2.17) or (2.6) for a sequence of standard BFGS updates, that are discussed in this paper. The multiple BFGS updates have the disadvantage, however, that B_k or H_k is not in general symmetric and positive definite even if the condition $s_i^T y_i > 0$, $i = 0, ..., k-1$, that guarantees that the matrix produced by k consecutive, standard BFGS updates is symmetric and positive definite, is satisfied. Instead, the multiple secant updates require the much stronger condition that $Y_k^T S_k$ be symmetric and positive definite, and there does not appear to be a practical way to enforce this condition computationally. Schnabel (1983) has instead considered ways to perturb Y_k to $\tilde{Y}_k$ so that $\tilde{Y}_k^T S_k$ is symmetric and positive definite, and the most recent secant condition (i.e. the last column of Y_k) is unchanged. In addition, if the columns of S_k are not strongly linear independent, the updates (7.3) or (7.4) may be numerical unstable so some secant pairs must be dropped from S_k and Y_k. Due to the additional computations required by these perturbations, and the lack of symmetry and positive definiteness in the unperturbed multiple secant BFGS update, it does not seem advantageous to use the multiple secant BFGS update rather than k consecutive, standard BFGS updates in the context of limited memory methods. An interesting related question is whether there is a natural perturbation of Y_k that causes the multiple secant update to be equivalent to (2.17); this does not seem to be the case, but as mentioned below the situation is different for the SR1 update.

Now we turn to the SR1 update. The multiple secant SR1 update, which to our knowledge was first discussed in Schnabel (1983), if applied to B_0 is given by

$$B_k = B_0 + (Y_k - B_0 S_k)((Y_k - B_0 S_k)^T S_k)^{-1}(Y_k - B_0 S_k)^T. \qquad (7.5)$$

The matrix B_k given by (7.5) always obeys the k secant equations $B_k S_k = Y_k$. Assuming B_0 is symmetric, B_k is symmetric if and only if $Y_k^T S_k$ is symmetric, which is true if $f(x)$ is quadratic but not necessarily otherwise. Like the standard SR1 update, B_k given by

(7.5) is not necessarily positive definite even if the necessary conditions for the standard BFGS or multiple BFGS update to be positive definite are met.

Comparing the multiple SR1 update (7.5) to the formula (5.2) for k consecutive, standard SR1 updates, it is clear that the only difference between these two formulae is that (7.5) contains the term $Y_k^T S_k$ as part of the middle, inverse expression, instead of the symmetric term $D_k + L_k + L_k^T$ in (5.2). Recalling that $Y_k^T S_k = \bar{R}_k^T + D_k + L_k^T$, it is immediate that (7.5) and (5.2) are identical if $\bar{R}_k = L_k$, i.e. if $s_i^T y_j = s_j^T y_i$ for all $0 \leq i, j \leq k-1$. This condition is true for $f(x)$ quadratic, and in this case the multiple SR1 update is the same as k consecutive, standard SR1 updates. This should come as no surprise, because the quadratic termination result for the standard SR1 update also implies that the update preserves all past secant equations, as does the multiple secant form of the SR1. Note that the condition for the equivalence of the multiple SR1 to k consecutive, standard SR1 updates is far milder than the assumption of conjugate directions required for the equivalence of k standard BFGS updates to the multiple BFGS in the quadratic case.

For non-quadratic $f(x)$, however, the standard and multiple SR1 updates will generally be different. Again, the algebraic costs associated with using the updates are very similar, while the multiple SR1 has the disadvantage that it does not, in general, preserve symmetry, while a sequence of standard SR1 updates does. Also, it is easier to monitor stability of the standard SR1, since this only involves considering each individual term $(y_j - B_j s_j)^T s_j$ rather than the matrix $(Y_k - B_0 S_k)^T S_k$. For this reason, a sequence of standard SR1 updates would seem preferable to the multiple SR1 update in the context of limited memory methods. It is interesting to note that if Y_k is perturbed to the $\tilde{Y}_k$ that one obtains by multiplying B_k given in (5.2) by S_k, then the multiple secant update becomes identical to (5.2). The same relationship is not true for the multiple BFGS update.

Finally we consider the Broyden update for nonlinear equations. A multiple secant version of Broyden's update has been considered by several authors including Barnes (1965), Gay and Schnabel (1978), and Schnabel (1983). In a limited context using the notation of Section 6, it is given by

$$A_k = A_0 + (Y_k - A_0 S_k)(S_k^T S_k)^{-1} S_k^T \tag{7.6}$$

This update is well defined as long as S_k has full column rank, and obeys the k secant equations $A_k S_k = Y_k$.

Comparing (7.6) to the formula (6.4) for k consecutive, standard Broyden updates, one sees that the only difference is in the matrix in the middle of the formula that is inverted. In the multiple secant update it is $S_k^T S_k$, while in (6.4) it is the upper triangular portion of this matrix, including the main diagonal. Therefore, the two updates are the same if the directions in S_k are orthogonal. The preference between these two formulas does not appear to be clear cut. The formula (6.4) has the advantage that it is well defined for any S_k, while (7.6) is only well defined numerically if the k step directions that make up S_k are sufficiently linearly independent. (If they are not, only some subset of them can be utilized in a numerical implementation of the multiple Broyden method; this is the approach that has been taken in implementations of this update.) On the other hand, (7.6) always enforces the k prior secant equations while (6.4) generally only enforces the most recent equation. Thus it would probably be worthwhile considering either method (or their inverse formulations) in a limited memory method for solving nonlinear equations. Note that the key difference between this comparison and the preceding comparisons of the BFGS and SR1 based formulae is that symmetry, which in general is inconsistent with satisfying multiple secant equations, is not a factor in the nonlinear equations case but is a factor for updates for optimization problems.

Acknowledgement. We would like to thank Peihuang Lu for considerable help in the preparation of this paper.

REFERENCES

J. Barnes, "An algorithm for solving nonlinear equations based on the secant method," *Computer Journal* 8 (1965) 66–67.

L. Biegler, J. Nocedal and C. Schmid, "Reduced Hessian methods for large scale constrained optimization," Technical Report, Department of Electrical Engineering and Computer Science, Northwestern University (Evanston, IL, 1993).

C.G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of Computation* 19 (1965) 577–593.

A. Buckley and A. LeNir, "QN-like variable storage conjugate gradients", *Mathematical Programming* 27, (1983) 103-119.

R.H. Byrd, P. Lu and J. Nocedal, "A limited memory algorithm for bound constrained optimization," Technical Report, Department of Electrical Engineering and Computer Science, Northwestern University (Evanston, IL, 1993).

A.R. Conn, N.I.M. Gould, and Ph.L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables," *Mathematics of Computation* 50/182 (1988) 399–430.

J.E. Dennis Jr. and R.B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations* (Prentice-Hall, 1983).

R. Fletcher, *Practical Methods of Optimization* (second edition) (John Wiley and Sons, Chichester, 1987).

D.M. Gay and R.B. Schnabel, "Solving systems of nonlinear equations by Broyden's method with projected updates," in: O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., *Nonlinear Programming* 3 (Academic Press, New York, 1978) 245–281.

J.C. Gilbert and C. Lemaréchal, "Some numerical experiments with variable storage quasi-Newton algorithms," *Mathematical Programming* 45 (1989) 407–436.

J.C. Gilbert and J. Nocedal, "The limited memory step computation and automatic differentiation," Technical Report NAM 02, Department of Electrical Engineering and Computer Science, Northwestern University (Evanston, IL, 1991), to appear in *Applied Math Letters*.

A. Griewank, "On automatic differentiation," in: M. Iri and K. Tanabe, eds., *Mathematical Programming* (Kluwer Academic Publishers, Tokyo, 1989) 83–107.

H. Fayez Khalfan, "Topics in quasi-Newton methods for unconstrained optimization," Ph.D. thesis, Department of Mathematics, University of Colorado (Boulder, CO, 1989).

H. Fayez Khalfan, R.H. Byrd, and R.B. Schnabel, "A theoretical and experimental study of the symmetric rank one update," *SIAM Journal on Optimization* 3 (1993) 1–24.

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

D.Q. Mahidhara and L. Lasdon, "An SQP algorithm for large sparse nonlinear programs," Technical report, MSIS Department, School of Business Administration, University of

Texas (Austin, TX, 1990).

H. Matthies and G. Strang, "The Solution of nonlinear finite element equations," *International Journal of Numerical Methods in Engineering* 14 (1979) 1613–1626.

J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation* 35 (1980) 773–782.

J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, 1970).

R.B. Schnabel, "Quasi-Newton methods using multiple secant equations," Technical Report CU-CS-247-83, Department of Computer Science, University of Colorado (Boulder, CO, 1983).

H.F. Walker, "Implementation of the GMRES method using Householder transformations", *SIAM Journal on Scientific and Statistical Computing* 9/1 (1988) 152–163.