SIAM J. OPTIM.
Vol. 12, No. 1, pp. 209–237
© 2001 Society for Industrial and Applied Mathematics

REDUCED-HESSIAN QUASI-NEWTON METHODS FOR UNCONSTRAINED OPTIMIZATION*

PHILIP E. GILL† AND MICHAEL W. LEONARD‡

Abstract. Quasi-Newton methods are reliable and efficient on a wide range of problems, but they can require many iterations if the problem is ill-conditioned or if a poor initial estimate of the Hessian is used. In this paper, we discuss methods designed to be more efficient in these situations. All the methods to be considered exploit the fact that quasi-Newton methods accumulate approximate second-derivative information in a sequence of expanding subspaces. Associated with each of these subspaces is a certain *reduced* approximate Hessian that provides a complete but compact representation of the second derivative information approximated up to that point. Algorithms that compute an explicit reduced-Hessian approximation have two important advantages over conventional quasi-Newton methods. First, the amount of computation for each iteration is significantly less during the early stages. This advantage is increased by forcing the iterates to *linger* on a manifold whose dimension can be significantly smaller than the subspace in which curvature has been accumulated. Second, approximate curvature along directions that lie off the manifold can be reinitialized as the iterations proceed, thereby reducing the influence of a poor initial estimate of the Hessian. These advantages are illustrated by extensive numerical results from problems in the CUTE test set. Our experiments provide strong evidence that reduced-Hessian quasi-Newton methods are more efficient and robust than conventional BFGS methods and some recently proposed extensions.

Key words. unconstrained optimization, quasi-Newton methods, BFGS method, conjugate-direction methods

AMS subject classifications. 65K05, 90C30

PII. S1052623400307950

1. Introduction.

Quasi-Newton methods are arguably the most effective methods for finding a minimizer of a smooth nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$ when second derivatives are either unavailable or too expensive to calculate. Quasi-Newton methods build up second-derivative information by estimating the curvature along a sequence of search directions. Each curvature estimate is installed in an approximate Hessian by applying a rank-one or rank-two update. One of the most successful updates is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula, which is a member of the wider Broyden class of rank-two updates (see section 2 for details).

Despite the success of these methods on a wide range of problems, it is well known that conventional quasi-Newton methods can require a disproportionately large number of iterations and function evaluations on some problems. This inefficiency may be caused by a poor choice of initial approximate Hessian or may result from the search direction's being poorly defined when the Hessian is ill-conditioned. This paper is concerned with the formulation of methods that are less susceptible to these difficulties.

All the methods to be discussed are based on exploiting an important property of quasi-Newton methods in which second-derivative information is accumulated in a

*Received by the editors January 19, 2000; accepted for publication (in revised form) March 5, 2001; published electronically October 18, 2001.

http://www.siam.org/journals/siopt/12-1/30795.html

†Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112 (pgill@ucsd.edu). This author's research was supported by National Science Foundation grant DMI-9424639 and Office of Naval Research grant N00014-96-1-0274.

‡Department of Mathematics, University of California, Los Angeles, CA 90095-1555 (na.mleonard@na-net.ornl.gov). This author's research was supported by National Science Foundation grant DMI-9424639.

sequence of expanding subspaces. At the kth iteration ($k < n$) curvature is known along vectors that lie in a certain *gradient subspace* whose dimension is no greater than $k+1$. This property is well documented in the context of solving positive-definite symmetric systems $Ax = b$. In particular, the iterates lie on an expanding sequence of manifolds characterized by the Krylov subspace associated with A (see, e.g., Freund, Golub, and Nachtigal [7] and Kelley [14, p. 12]). These manifolds are identical to those associated with the BFGS method applied to minimizing the quadratic $c - b^T x + \frac{1}{2} x^T A x$. (For further details of the equivalence of quasi-Newton methods and conjugate-gradient methods, see Nazareth [22].)

In the quasi-Newton context, the availability of an explicit basis for the gradient subspace makes it possible to represent the approximate curvature in terms of a *reduced* approximate Hessian matrix with order at most $k + 1$. Quasi-Newton algorithms that explicitly calculate a reduced Hessian have been proposed by Fenelon [4] and Nazareth [21], who also considered modified Newton methods in the same context. Siegel [27] has proposed methods that work with a reduced inverse approximate Hessian. In practical terms, the reduced-Hessian formulation can require significantly less work per iteration when k is small relative to n. This property can be exploited by forcing iterates to *linger* on a manifold while the objective function is minimized to greater accuracy. While iterates linger, the search direction is calculated from a system that is generally smaller than the reduced Hessian. In many practical situations convergence occurs before the dimension of the lingering subspace reaches n, resulting in substantial savings in computing time (see section 7).

More recently, Siegel [28] has proposed the conjugate-direction scaling algorithm, which is a quasi-Newton method based on a conjugate-direction factorization of the inverse approximate Hessian. Although no explicit reduced Hessian is updated, the method maintains a basis for the expanding subspaces and allows iterates to linger on a manifold. The method also has the benefit of finite termination on a quadratic (see Leonard [16, p. 77]). More importantly, Siegel's method includes a feature that can considerably enhance the benefits of lingering. Siegel notes that the search direction is the sum of two vectors: one with the scale of the estimated derivatives and the other with the scale of the initial approximate Hessian. Siegel suggests rescaling the second vector using newly computed approximate curvature. Algorithms that combine lingering and rescaling have the potential for giving significant improvements over conventional quasi-Newton methods. Lingering forces the iterates to remain on a manifold until the curvature has been sufficiently established; rescaling ensures that the initial curvature in the unexplored manifold is commensurate with curvature already found.

In this paper we propose several algorithms based on maintaining the triangular factors of an explicit reduced Hessian. We demonstrate how these factors can be used to force the iterates to linger while curvature information continues to be accumulated along directions lying off the manifold. With the BFGS method, it is shown that while lingering takes place, the new curvature is restricted to an upper-trapezoidal portion of the factor of the reduced Hessian and the remaining portion retains the diagonal structure of the initial approximate Hessian. It follows that conjugate-direction scaling is equivalent to simply *reinitializing* the diagonal part of the reduced Hessian with freshly computed curvature information.

Despite the similarities between reduced-Hessian reinitialization and conjugate-direction scaling, it must be emphasized that these methods are not the same, in the sense that they involve very different storage and computational overheads. More-

over, the reduced-Hessian factorization has both practical and theoretical advantages over Siegel's conjugate-direction factorization. On the practical side, the early search directions can be calculated with significantly less work. This can result in a significantly faster minimization when the dimension of the manifold grows relatively slowly, as it does on many problems (see sections 6 and 7). On the theoretical side, the simple structure exhibited by the reduced-Hessian factor allows the benefits of reinitialization to be extended to the large-scale case (see Gill and Leonard [9]).

A reduced-Hessian method allows *expansion* of the manifold on which curvature information is known. Thus, when implementing software, it is necessary either to allocate new memory dynamically as the reduced Hessian grows or to reserve sufficient storage space in advance. In practice, however, the order of the reduced Hessian often remains much less than n, i.e., the problem is solved without needing room for an $n \times n$ matrix. Notwithstanding this benefit, on very large problems it may be necessary to *explicitly* limit the amount of storage used, by placing a limit on the order of the reduced Hessian. Such *limited-memory* reduced-Hessian methods discard old curvature information whenever the addition of new information causes a predefined storage limit to be exceeded. Methods of this type have been considered by Fenelon [4] and Siegel [27]. Limited-memory methods directly related to the methods considered in this paper are discussed by Gill and Leonard [9].

The paper is organized as follows. Section 2 contains a discussion of various theoretical aspects of reduced-Hessian quasi-Newton methods, concluding with the statement of Algorithm RH, a reduced-Hessian formulation of a conventional quasi-Newton method. Algorithm RH is the starting point for the improved algorithms presented in sections 3 and 4. Section 3 is concerned with the effects of lingering on the form of the factorization of the reduced Hessian. In section 4, Siegel's conjugate-direction scaling algorithm is reformulated as an explicit reduced-Hessian method. In section 4.1 we present a reduced-Hessian method that combines lingering with reinitialization. The convergence properties of this algorithm are discussed in sections 4.2 and 4.3. To simplify the discussion, the algorithms of sections 2–4 are given with the assumption that all computations are performed in exact arithmetic. The effects of rounding error are discussed in section 5. Finally, sections 6 and 7 include some numerical results when various reduced-Hessian algorithms are applied to test problems taken from the CUTE test collection (see Bongartz et al. [1]). Section 6 also includes comparisons with Siegel's method and with Lalee and Nocedal's automatic column-scaling method [15], which is another extension of the BFGS method. Results from the package NPSOL [10] are provided to illustrate how the reduced-Hessian approach compares to a conventional quasi-Newton method. Our experiments demonstrate that reduced-Hessian methods can require substantially less computer time than these alternatives. Part of the reduction in computer time corresponds to the smaller number of iterations and function evaluations required when using the reinitialization strategy. However, much of this reduction comes from the fact that the average cost of an iteration is less than for competing methods.

Unless explicitly indicated otherwise, $\|\cdot\|$ denotes the vector two-norm or its subordinate matrix norm. The vector e_i is used to denote the ith unit vector of the appropriate dimension. A floating-point operation, or *flop*, refers to a calculation of the form $\alpha x + y$, i.e., a multiplication and an addition.

2. Background. Given a twice-continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ with gradient vector ∇f and Hessian matrix $\nabla^2 f$, the kth iteration of a quasi-Newton

method is given by

$$H_k p_k = -\nabla f(x_k), \qquad x_{k+1} = x_k + \alpha_k p_k, \tag{2.1}$$

where H_k is a symmetric, positive-definite matrix, p_k is the search direction, and α_k is a scalar step length. If H_k is interpreted as an approximation to $\nabla^2 f(x_k)$, then $x_k + p_k$ can be viewed as minimizing a quadratic model of f with Hessian H_k. The matrix H_{k+1} is obtained from H_k by adding a low-rank matrix defined in terms of $\delta_k = x_{k+1} - x_k$ and $\gamma_k = g_{k+1} - g_k$, where $g_k = \nabla f(x_k)$. Updates from the Broyden class give a matrix H_{k+1} such that

$$H_{k+1} = H_k - \frac{1}{\delta_k^T H_k \delta_k} H_k \delta_k \delta_k^T H_k + \frac{1}{\gamma_k^T \delta_k} \gamma_k \gamma_k^T + \phi_k (\delta_k^T H_k \delta_k) w_k w_k^T, \tag{2.2}$$

where $w_k = \gamma_k / \gamma_k^T \delta_k - H_k \delta_k / \delta_k^T H_k \delta_k$, and ϕ_k is a scalar parameter. It is generally accepted that the most effective update corresponds to $\phi_k = 0$, which defines the well-known BFGS update

$$H_{k+1} = H_k - \frac{1}{\delta_k^T H_k \delta_k} H_k \delta_k \delta_k^T H_k + \frac{1}{\gamma_k^T \delta_k} \gamma_k \gamma_k^T. \tag{2.3}$$

For brevity, the term "Broyden's method" refers to a method based on iteration (2.1) when used with an update from the Broyden class. Similarly, the term "BFGS method" refers to iteration (2.1) with the BFGS update.

The scalar $\gamma_k^T \delta_k$, known as the *approximate curvature*, is a difference estimate of the (unknown) curvature $\delta_k^T \nabla^2 f(x_k) \delta_k$. Each Broyden update gives an approximate Hessian satisfying $\delta_k^T H_{k+1} \delta_k = \gamma_k^T \delta_k$, which implies that the approximate curvature $\gamma_k^T \delta_k$ is installed as the *exact* curvature of the new quadratic model in the direction δ_k. It follows that a positive value for the approximate curvature is a necessary condition for H_{k+1} to be positive definite.

We follow common practice and restrict our attention to Broyden updates with the property that if H_k is positive definite, then H_{k+1} is positive definite if and only if $\gamma_k^T \delta_k > 0$. This restriction allows H_{k+1} to be kept positive definite by using a step length algorithm that ensures a positive value of the approximate curvature. Practical step length algorithms also include a criterion for sufficient descent. Two criteria often used are the Wolfe conditions

$$f(x_k + \alpha_k p_k) \leq f(x_k) + \mu \alpha_k g_k^T p_k \quad \text{and} \quad g_{k+1}^T p_k \geq \eta g_k^T p_k, \tag{2.4}$$

where the constants μ and η are chosen so that $0 \leq \mu \leq \eta < 1$ and $\mu < \frac{1}{2}$.

If n is sufficiently small that an $n \times n$ dense matrix can be stored explicitly, two alternative methods have emerged for implementing quasi-Newton methods. The first is based on using the upper-triangular matrix C_k such that $H_k = C_k^T C_k$ (see Gill et al. [10]). The second uses a matrix V_k satisfying the conjugate-direction identity $V_k^T H_k V_k = I$ (see Powell [25], Siegel [28]). Neither of these methods store H_k (or its inverse) as an explicit matrix. Instead, C_k or V_k is updated directly by exploiting the fact that every update from the Broyden class defines a rank-one update to C_k or V_k (see Goldfarb [12] and Dennis and Schnabel [3]). The rank-one update to C_k generally destroys the upper-triangular form of C_k. However, the updated C_k can be restored to upper-triangular form in $\mathcal{O}(n^2)$ operations.

2.1. Reduced-Hessian quasi-Newton methods. In this section, we review the formulation of conventional quasi-Newton methods as reduced-Hessian methods. The next key result is proved by Siegel [27] (see Fletcher and Powell [6], and Fenelon [4] for similar results in terms of the DFP and BFGS updates). Let $\mathcal{G}_k$ denote the subspace $\mathcal{G}_k = \text{span}\{g_0, g_1, \ldots, g_k\}$, and let $\mathcal{G}_k^\perp$ denote the orthogonal complement of $\mathcal{G}_k$ in $\mathbb{R}^n$.

LEMMA 2.1. *Consider the Broyden method applied to a general nonlinear function. If $H_0 = \sigma I$ ($\sigma > 0$), then $p_k \in \mathcal{G}_k$ for all k. Moreover, if $z \in \mathcal{G}_k$ and $w \in \mathcal{G}_k^\perp$, then $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$.* □

Let r_k denote $\dim(\mathcal{G}_k)$, and let B_k (B for "basis") denote an $n \times r_k$ matrix whose columns form a basis for $\mathcal{G}_k$. An orthonormal basis matrix Z_k can be defined from the QR decomposition $B_k = Z_k T_k$, where T_k is a nonsingular upper-triangular matrix.[1] Let the $n - r_k$ columns of W_k define an orthonormal basis for $\mathcal{G}_k^\perp$. If Q_k is the orthogonal matrix $Q_k = \begin{pmatrix} Z_k & W_k \end{pmatrix}$, then the transformation $x = Q_k x_Q$ defines a transformed approximate Hessian $Q_k^T H_k Q_k$ and a transformed gradient $Q_k^T g_k$. If $H_0 = \sigma I$ ($\sigma > 0$), it follows from (2.2) and Lemma 2.1 that the transformation induces a block-diagonal structure, with

$$Q_k^T H_k Q_k = \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma I_{n-r_k} \end{pmatrix} \quad \text{and} \quad Q_k^T g_k = \begin{pmatrix} Z_k^T g_k \\ 0 \end{pmatrix}. \tag{2.5}$$

The positive-definite matrix $Z_k^T H_k Z_k$ is known as a *reduced* approximate Hessian (or just reduced Hessian). The vector $Z_k^T g_k$ is known as a reduced gradient.

If we write the equation for the search direction as $(Q_k^T H_k Q_k) Q_k^T p_k = -Q_k^T g_k$, it follows from (2.5) that

$$p_k = Z_k q_k, \quad \text{where } q_k \text{ satisfies } Z_k^T H_k Z_k q_k = -Z_k^T g_k. \tag{2.6}$$

If the Cholesky factorization $Z_k^T H_k Z_k = R_k^T R_k$ is known, q_k can be computed from the forward substitution $R_k^T d_k = -Z_k^T g_k$ and back-substitution $R_k q_k = d_k$. A benefit of this approach is that Z_k and R_k require less storage than H_k when $k \ll n$ (see Gill and Leonard [9]). In addition, the computation of p_k when $k \ll n$ requires less work than it does for methods that store C_k or V_k. A benefit of using an orthonormal Z_k is that $\text{cond}(Z_k^T H_k Z_k) \leq \text{cond}(H_k)$, where $\text{cond}(\cdot)$ denotes the spectral condition number (see, e.g., Gill, Murray, and Wright [11, p. 162]).

There are a number of alternative choices for the basis B_k. Both Fenelon and Siegel propose that B_k be formed from a linearly independent subset of $\{g_0, g_1, \ldots, g_k\}$. With this choice, the orthonormal basis can be accumulated columnwise as the iterations proceed using Gram–Schmidt orthogonalization (see, e.g., Golub and Van Loan [13, pp. 218–220]). During iteration k, the number of columns of Z_k either remains unchanged or increases by one, depending on the value of the scalar ρ_{k+1}, such that $\rho_{k+1} = \|(I - Z_k Z_k^T) g_{k+1}\|$. If $\rho_{k+1} = 0$, the new gradient has no component outside range(Z_k) and g_{k+1} is said to be *rejected*. Thus, if $\rho_{k+1} = 0$, then Z_k already provides a basis for $\mathcal{G}_{k+1}$ with $r_{k+1} = r_k$ and $Z_{k+1} = Z_k$. Otherwise, $r_{k+1} = r_k + 1$ and the gradient g_{k+1} is said to be *accepted*. In this case, Z_k gains a new column z_{k+1} defined by the identity $\rho_{k+1} z_{k+1} = (I - Z_k Z_k^T) g_{k+1}$. The calculation of z_{k+1} also provides the r_k-vector $u_k = Z_k^T g_{k+1}$ and the scalar $z_{k+1}^T g_{k+1}$ ($= \rho_{k+1}$), which are the components of the reduced gradient $Z_{k+1}^T g_{k+1}$ for the next iteration. This orthogonalization procedure requires approximately $2nr_k$ flops.

[1] The matrix T_k appears only in the theoretical discussion—it is not needed for computation.

Definition (2.6) of each search direction implies that $p_j \in \mathcal{G}_k$ for all $0 \leq j \leq k$. This leads naturally to another basis for $\mathcal{G}_k$ based on orthogonalizing the search directions $p_0, p_1, \ldots, p_k$. The next lemma implies that Z_k can be defined not only by the accepted gradients, but also by the corresponding search directions.

LEMMA 2.2. *At the start of iteration k, let Z_k denote the matrix obtained by orthogonalizing the gradients $g_0, g_1, \ldots, g_k$ of Broyden's method. Let P_k and G_k denote the matrices of search directions and gradients associated with iterations at which a gradient is accepted. Then there are nonsingular upper-triangular matrices T_k and $\widehat{T}_k$ such that $G_k = Z_k T_k$ and $P_k = Z_k \widehat{T}_k$.*

Proof. Without loss of generality, it is assumed that every gradient is accepted. The proof is by induction on the iteration number k.

The result is true for $k = 0$ because the single column $g_0/\|g_0\|$ of Z_0 is identical to the normalized version of the search direction $p_0 = -g_0/\sigma$.

If the result is true at the start of iteration $k-1$, there exist nonsingular T_{k-1} and $\widehat{T}_{k-1}$ with $G_{k-1} = Z_{k-1}T_{k-1}$ and $P_{k-1} = Z_{k-1}\widehat{T}_{k-1}$. At the start of iteration k, the last column of Z_k satisfies $\rho_k z_k = g_k - Z_{k-1}Z_{k-1}^T g_k$, and

$$G_k = \begin{pmatrix} G_{k-1} & g_k \end{pmatrix} = \begin{pmatrix} Z_{k-1} & z_k \end{pmatrix} \begin{pmatrix} T_{k-1} & Z_{k-1}^T g_k \\ 0 & \rho_k \end{pmatrix} = Z_k T_k. \tag{2.7}$$

The last equality defines T_k, which is nonsingular since $\rho_k \neq 0$. Since $p_k = Z_k Z_k^T p_k$, we find

$$P_k = \begin{pmatrix} P_{k-1} & p_k \end{pmatrix} = \begin{pmatrix} Z_{k-1} & z_k \end{pmatrix} \begin{pmatrix} \widehat{T}_{k-1} & Z_{k-1}^T p_k \\ 0 & z_k^T p_k \end{pmatrix} = Z_k \widehat{T}_k,$$

where the last equality defines $\widehat{T}_k$. The scalar $z_k^T p_k$ is nonzero (see Leonard [16, pp. 94–99][2]), which implies that $\widehat{T}_k$ is nonsingular, and thus the induction is complete. ∎

Lemma 2.2 can be used to show that Z_k provides an orthonormal basis for the span $\mathcal{P}_k$ of *all* search directions $\{p_0, p_1, \ldots, p_k\}$.

THEOREM 2.3. *The subspaces $\mathcal{G}_k$ and $\mathcal{P}_k$ generated by the gradients and search directions of the conventional Broyden method are identical.*

Proof. The definition of each p_j $(0 \leq j \leq k)$ implies that $\mathcal{P}_k \subseteq \mathcal{G}_k$. Lemma 2.2 implies that $\mathcal{G}_k = \text{range}(P_k)$. Since $\text{range}(P_k) \subseteq \mathcal{P}_k$, it follows that $\mathcal{G}_k \subseteq \mathcal{P}_k$. ∎

Given Z_{k+1} and H_{k+1}, the calculation of the search direction for the next iteration requires the Cholesky factor of $Z_{k+1}^T H_{k+1} Z_{k+1}$. This factor can be obtained from R_k in a two-step process that does not require knowledge of H_k. The first step, which is not needed if g_{k+1} is rejected, is to compute the factor R_k'' of $Z_{k+1}^T H_k Z_{k+1}$ (the symbol R_k' is reserved for use in section 3). This step involves adding a row and column to R_k to account for the new last column of Z_{k+1}. It follows from Lemma 2.1 and (2.5) that

$$Z_{k+1}^T H_k Z_{k+1} = \begin{pmatrix} Z_k^T H_k Z_k & Z_k^T H_k z_{k+1} \\ z_{k+1}^T H_k Z_k & z_{k+1}^T H_k z_{k+1} \end{pmatrix} = \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma \end{pmatrix},$$

[2]The proof is nontrivial and is omitted for brevity.

giving an expanded block-diagonal factor R_k'' defined by

$$R_k'' = \begin{cases} R_k, & \text{if } r_{k+1} = r_k, \\ \begin{pmatrix} R_k & 0 \\ 0 & \sigma^{1/2} \end{pmatrix}, & \text{if } r_{k+1} = r_k + 1. \end{cases} \tag{2.8}$$

This expansion procedure involves the vectors $v_k = Z_k^T g_k$, $u_k = Z_k^T g_{k+1}$, and $q_k = Z_k^T p_k$, which are stored and updated for efficiency. As both p_k and g_k lie in Range(Z_k), if g_{k+1} is accepted, the vectors $v_k'' = Z_{k+1}^T g_k$ and $q_k'' = Z_{k+1}^T p_k$ are trivially defined from v_k and q_k by appending a zero component (cf. (2.5)). Similarly, the vector $u_k'' = Z_{k+1}^T g_{k+1}$ is formed from u_k and ρ_{k+1}. If g_{k+1} is rejected, then $v_k'' = v_k$, $u_k'' = u_k$ and $q_k'' = q_k$. In either case, v_{k+1} is equal to u_k'' and need not be calculated at the start of iteration $k+1$ (see Algorithm 2.1 below).

The second step of the modification alters R_k'' to reflect the rank-two quasi-Newton update to H_k. This update gives a modified factor $R_{k+1} = \boldsymbol{Broyden}(R_k'', s_k, y_k)$, where $s_k = Z_{k+1}^T(x_{k+1} - x_k) = \alpha_k q_k''$ and $y_k = Z_{k+1}^T(g_{k+1} - g_k) = u_k'' - v_k''$. The work required to compute R_{k+1} depends on the choice of Broyden update and the numerical method used to calculate $\boldsymbol{Broyden}(R_k'', s_k, y_k)$. For the BFGS update, R_{k+1} is the triangular factor associated with the QR factorization of $R_k'' + w_1 w_2^T$, where w_1 and w_2 are given by

$$w_1 = \frac{1}{\|R_k'' s_k\|} R_k'' s_k \quad \text{and} \quad w_2 = \frac{1}{(y_k^T s_k)^{1/2}}\, y_k - \frac{1}{\|R_k'' s_k\|} R_k''^T R_k'' s_k \tag{2.9}$$

(see Goldfarb [12] and Dennis and Schnabel [3]). R_{k+1} can be computed from R_k'' in $4r_k^2 + \mathcal{O}(r_k)$ flops using conventional plane rotations, or in $3r_k^2 + \mathcal{O}(r_k)$ flops using a modified rotation[3] (see Gill et al. [8]). These estimates exclude the cost of forming w_1 and w_2. The vector w_1 is computed in $O(r_k)$ operations from the vector $d_k/\|d_k\|$, where $d_k = -R_k^{-T} v_k$ is the intermediate quantity used in the calculation of q_k (see section 2.1). Similarly, w_2 is obtained in $O(r_k)$ operations using the identity $R_k''^T R_k'' s_k = -\alpha_k v_k''$ implied by (2.6) and the definition of $Z_{k+1}^T g_k$.

2.2. A reduced-Hessian method. We conclude this section by giving a complete reduced-Hessian formulation of a quasi-Newton method from the Broyden class. This method involves two main procedures: an *expand*, which determines Z_{k+1} using the Gram–Schmidt QR process and possibly increases the order of the reduced Hessian by one; and an *update*, which applies a Broyden update directly to the reduced Hessian. For brevity, we use the expression $(Z_{k+1}, R_k'', u_k'', v_k'', q_k'', r_{k+1}) = \boldsymbol{expand}(Z_k, R_k, u_k, v_k, q_k, g_{k+1}, r_k, \sigma)$ to signify the input and output quantities associated with the expand procedure. This statement should be interpreted as the following: Given values of the quantities Z_k, R_k, u_k, v_k, q_k, g_{k+1}, r_k, and σ, the expand procedure computes values of Z_{k+1}, R_k'', u_k'', v_k'', q_k'', r_{k+1}. (Unfortunately, the need to associate quantities used in the algorithm with quantities used in its derivation leads to an algorithm that is more complicated than its computer implementation. In practice, almost all most quantities are updated *in situ*.)

[3]Certain special techniques can be used to reduce this flop count further; see Goldfarb [12].

ALGORITHM 2.1. REDUCED-HESSIAN QUASI-NEWTON METHOD (RH).

Choose x_0 and σ ($\sigma > 0$);
$k = 0$; $r_0 = 1$; $g_0 = \nabla f(x_0)$;
$Z_0 = g_0/\|g_0\|$; $R_0 = \sigma^{1/2}$; $v_0 = \|g_0\|$;
while not converged **do**
  Solve $R_k^T d_k = -v_k$; $R_k q_k = d_k$;
  $p_k = Z_k q_k$;
  Find α_k satisfying the Wolfe conditions (2.4);
  $x_{k+1} = x_k + \alpha_k p_k$; $g_{k+1} = \nabla f(x_k + \alpha_k p_k)$; $u_k = Z_k^T g_{k+1}$;
  $(Z_{k+1}, r_{k+1}, R_k'', u_k'', v_k'', q_k'') = \boldsymbol{expand}(Z_k, r_k, R_k, u_k, v_k, q_k, g_{k+1}, \sigma)$;
  $s_k = \alpha_k q_k''$; $y_k = u_k'' - v_k''$; $R_{k+1} = \boldsymbol{Broyden}(R_k'', s_k, y_k)$;
  $v_{k+1} = u_k''$; $k \leftarrow k + 1$;
end do

In exact arithmetic, Algorithm RH generates the same iterates as its conventional Broyden counterpart, and the methods differ only in the storage needed and the number of operations per iteration. Since v_k is defined as a by-product of the orthogonalization, the computation of p_k involves the solution of two triangular systems and a matrix-vector product, requiring a total of approximately $nr_k + r_k^2$ flops. For the BFGS update, $3r_k^2 + \mathcal{O}(r_k)$ flops are required to update R_k, with the result that Algorithm RH requires approximately $(3r_k + 1)n + 4r_k^2 + \mathcal{O}(r_k)$ flops for each BFGS iteration. As r_k increases, the flop count approaches $7n^2 + \mathcal{O}(n)$. When r_k reaches n, Z_k is full and no more gradients are accepted; only $Z_k^T g_{k+1}$ is computed during the orthogonalization, and the work drops to $6n^2 + \mathcal{O}(n)$. Although H_k is not stored explicitly, it is always implicitly defined by reversing (2.5), i.e.,

$$H_k = Q_k R_Q^T R_Q Q_k^T, \quad \text{where} \quad R_Q = \begin{pmatrix} R_k & 0 \\ 0 & \sigma^{1/2} I_{n-r_k} \end{pmatrix} \tag{2.10}$$

and $Q_k = \begin{pmatrix} Z_k & W_k \end{pmatrix}$.

2.3. Geometric considerations. Next we consider the application of Algorithm RH to the strictly convex quadratic

$$f(x) = c - b^T x + \tfrac{1}{2} x^T A x, \tag{2.11}$$

where c is a scalar, b is an n-vector, and A is an $n \times n$ constant symmetric positive-definite matrix. Suppose that the BFGS update is used, and that each α_k is computed from an exact line search (i.e., α_k minimizes $f(x_k + \alpha p_k)$ with respect to α). Under these circumstances, it can be shown that the $k+1$ columns of Z_k are the normalized gradients $\{g_i/\|g_i\|\}$, and that R_k is upper bidiagonal with nonzero components $r_{ii} = \|g_{i-1}\|/(y_{i-1}^T s_{i-1})^{1/2}$ and $r_{i,i+1} = -\|g_i\|/(y_{i-1}^T s_{i-1})^{1/2}$ for $1 \le i \le k$, and $r_{k+1,k+1} = \sigma^{1/2}$ (see Fenelon [4]). These relations imply that the search directions satisfy

$$p_0 = -\frac{1}{\sigma} g_0, \qquad p_k = -\frac{1}{\sigma} g_k + \beta_{k-1} p_{k-1}, \qquad k \ge 1,$$

with $\beta_{k-1} = \|g_k\|^2/\|g_{k-1}\|^2$. These vectors are parallel to the well-known conjugate-gradient search directions (cf. Corollary 4.2). When used with an exact line search, the search directions and gradients satisfy the relations (i) $p_i^T A p_j = 0$, $i \ne j$; (ii) $g_i^T g_j = 0$, $i \ne j$; and (iii) $g_i^T p_i = -\|g_i\|^2/\sigma$ (see, e.g., Fletcher [5, p. 81] for a proof for

the case $\sigma = 1$). The identities (i), (ii), and (iii) can be used to show that if the search directions are independent, then the local quadratic model $\varphi(p) = g_k^T p + \frac{1}{2} p^T H_k p$ is exact at the start of iteration $n+1$, i.e., $H_{n+1} = A$.

Since the columns of Z_k are the normalized gradients, the BFGS orthogonality relations (ii) imply that a new gradient g_{k+1} can be rejected only if $g_{k+1} = 0$, at which point the algorithm terminates. It follows that the reduced Hessian steadily expands as the iterations proceed. The curvature of the local quadratic model $\varphi(p)$ along any unit vector in Range(W_k) depends only on the choice of H_0 and has no effect on the definition of p_k. Only curvature along directions in Range(Z_k) affects the definition of p_k, and this curvature is completely determined by $Z_k^T H_k Z_k$.

The next lemma implies that $f(x)$ is minimized on a sequence of expanding linear manifolds and that, at the start of iteration k, the curvature of the quadratic model is exact on a certain subspace of dimension k. Let $\mathcal{M}(\mathcal{G}_k)$ denote the linear manifold $\mathcal{M}(\mathcal{G}_k) = \{x_0 + z \mid z \in \mathcal{G}_k\}$ determined by x_0 and $\mathcal{G}_k$.

LEMMA 2.4. *Suppose that the BFGS method with an exact line search is applied to the strictly convex quadratic $f(x)$ (2.11). If $H_0 = \sigma I$, then at the start of iteration k,* (a) *x_k minimizes $f(x)$ on the linear manifold $\mathcal{M}(\mathcal{G}_{k-1})$, and* (b) *the curvature of the quadratic model is exact on the k-dimensional subspace $\mathcal{G}_{k-1}$. Thus, $z^T H_k z = z^T A z$ for all $z \in \mathcal{G}_{k-1}$.*

Proof. Part (a) follows directly from the identity $Z_{k-1}^T g_k = 0$ implied by the orthogonality of the gradients and the special form of Z_{k-1}.

To verify part (b), we write the normalized gradients in terms of the search directions. With Fenelon's form of Z_k and R_k, we find that $Z_k = -P_k D_k R_k$, where $P_k = \begin{pmatrix} p_0 & p_1 & \cdots & p_k \end{pmatrix}$ and D_k is the nonnegative diagonal matrix such that

$$D_k^2 = \sigma^2 \operatorname{diag}\left(\frac{y_0^T s_0}{\|g_0\|^4},\ \frac{y_1^T s_1}{\|g_1\|^4},\ \ldots,\ \frac{y_{k-1}^T s_{k-1}}{\|g_{k-1}\|^4},\ \frac{1}{\sigma \|g_k\|^2} \right).$$

A simple computation using the conjugacy condition (i) above gives the reduced Hessian as $Z_k^T A Z_k = R_k^T D_k P_k^T A P_k D_k R_k = R_k^T \widehat{D} R_k$, where

$$\widehat{D}_k = \sigma^2 \operatorname{diag}\left(\frac{y_0^T s_0}{\|g_0\|^4} p_0^T A p_0,\ \frac{y_1^T s_1}{\|g_1\|^4} p_1^T A p_1,\ \ldots,\ \frac{y_{k-1}^T s_{k-1}}{\|g_{k-1}\|^4} p_{k-1}^T A p_{k-1},\ \frac{p_k^T A p_k}{\sigma \|g_k\|^2} \right).$$

The definition of α_i as the minimizer of $f(x_i + \alpha p_i)$ implies that $\alpha_i = -g_i^T p_i / (p_i^T A p_i)$ and $g_{i+1}^T p_i = 0$. Hence, for all i such that $0 \le i \le k-1$, it follows that

$$y_i^T s_i = \alpha_i y_i^T p_i = -\alpha_i g_i^T p_i = (g_i^T p_i)^2 / p_i^T A p_i.$$

Using these identities with $g_i^T p_i = -\|g_i\|^2/\sigma$ from (iii) above, the expression for $\widehat{D}_k$ simplifies, with $\widehat{D}_k = \operatorname{diag}(I_k,\ 1/\alpha_k)$.

Finally, if Z_k is partitioned so that $Z_k = \begin{pmatrix} Z_{k-1} & g_k/\|g_k\| \end{pmatrix}$, where Z_{k-1} has k columns, then comparison of the leading $k \times k$ principal minors of the matrices $R_k^T R_k$ $(= Z_k^T H_k Z_k)$ and $R_k^T \widehat{D}_k R_k$ $(= Z_k^T A Z_k)$ gives the required identity $Z_{k-1}^T H_k Z_{k-1} = Z_{k-1}^T A Z_{k-1}$. ∎

Part (a) of this result allows us to interpret each new iterate x_{k+1} as "stepping onto" a larger manifold $\mathcal{M}(\mathcal{G}_k)$ such that $\mathcal{M}(\mathcal{G}_{k-1}) \subset \mathcal{M}(\mathcal{G}_k)$. This interpretation also applies when minimizing a general nonlinear function, as long as g_k is accepted for the definition of $\mathcal{G}_k$. (Recall from the proof of Lemma 2.2 that $z_k^T p_k \neq 0$ in this case.)

We say that the curvature along z is *established* if $z^T H_k z = z^T \nabla^2 f(x_k) z$. In particular, under the conditions of Lemma 2.4, the curvature is established at iteration k on all of Range($\mathcal{G}_{k-1}$).

3. Lingering on a manifold. Up to this point we have considered reduced-Hessian methods that generate the same iterates as their Broyden counterparts. Now we expand our discussion to include methods that are not necessarily equivalent to a conventional quasi-Newton method. Our aim is to derive methods with better robustness and efficiency.

When f is a general nonlinear function, the step from x_k to x_{k+1} is unlikely to minimize f on the manifold $\mathcal{M}(\mathcal{G}_k)$. However, in a sequence of iterations in which the gradient is rejected, Z_k remains constant, and the algorithm proceeds to minimize f on the manifold $\mathcal{M}(\mathcal{G}_k)$. In this section, we propose an algorithm in which iterates can remain, or "linger," on a manifold even though new gradients are being accepted. The idea is to linger on a manifold as long as a good reduction in f is being achieved. Lingering has the advantage that the order of the relevant submatrix of the reduced Hessian can be significantly smaller than that of the reduced Hessian itself.

An algorithm that can linger uses one of two alternative search directions: an *RH direction* or a *lingering direction*. An RH direction is defined as in Algorithm RH, i.e., an RH direction lies in $\mathcal{G}_k$ and is computed using the reduced Hessian associated with Z_k. As discussed above, an RH direction defines an x_{k+1} on the manifold $\mathcal{M}(\mathcal{G}_k)$. By contrast, a lingering direction forces x_{k+1} to remain on a manifold $\mathcal{M}(\mathcal{U}_k)$, such that $\mathcal{U}_k \subset \mathcal{G}_k$. Given a point $x_k \in \mathcal{M}(\mathcal{U}_k)$, the next iterate x_{k+1} will also lie on $\mathcal{M}(\mathcal{U}_k)$ as long as $p_k \in \mathcal{U}_k$. Accordingly, an algorithm is said to "linger on $\mathcal{M}(\mathcal{U}_k)$" if the search direction satisfies $p_k \in$ Range(U_k), where the columns of U_k form a basis for $\mathcal{U}_k$. As long as $\mathcal{U}_k$ remains constant and p_k has the form $p_k = U_k p_U$ for some p_U, the iterates $x_{k+1}, x_{k+2}, \ldots$ will continue to linger on $\mathcal{M}(\mathcal{U}_k)$.

The subspace $\mathcal{U}_k$ and an appropriate basis U_k are defined as follows. At the start of iteration k, an orthonormal basis for $\mathcal{G}_k$ is known such that

$$Z_k = \begin{pmatrix} U_k & Y_k \end{pmatrix}, \tag{3.1}$$

where U_k is an $n \times l_k$ matrix whose columns span the subspace $\mathcal{U}_k$ of all RH directions computed so far, and Y_k corresponds to a certain subset of the accepted gradients defined below. The integer l_k ($0 \le l_k \le r_k$) is known as the *partition parameter* for Z_k. It must be emphasized that the partition (3.1) is defined at *every* iteration, regardless of whether or not lingering occurs. The partition is necessary because quantities computed from U_k and Y_k are used to decide between an RH direction and a lingering direction.

The matrix Z_k is an orthonormal factor of a particular basis for $\mathcal{G}_k$ consisting of both gradients *and* search directions. Let P_k denote the $n \times l_k$ matrix of RH search directions computed so far, and let G_k denote an $n \times (r_k - l_k)$ matrix whose columns are a subset of the accepted gradients. (Note that the definitions of P_k and G_k are different from those of Lemma 2.2.) The matrix Z_k is the orthonormal factor corresponding to the QR factorization of the basis matrix $B_k = \begin{pmatrix} P_k & G_k \end{pmatrix}$, i.e., $\begin{pmatrix} P_k & G_k \end{pmatrix} = Z_k T_k$ for some nonsingular upper-triangular matrix T_k. If T_k is partitioned appropriately, we have

$$B_k = \begin{pmatrix} P_k & G_k \end{pmatrix} = Z_k T_k = \begin{pmatrix} U_k & Y_k \end{pmatrix} \begin{pmatrix} T_U & T_{UY} \\ 0 & T_Y \end{pmatrix}, \tag{3.2}$$

where T_U is an $l_k \times l_k$ upper-triangular matrix. Note that the definition of B_k implies that $\text{Range}(P_k) = \text{Range}(U_k T_U) = \text{Range}(U_k)$, as required.

Although the dimension of U_k remains fixed while iterates linger, the column dimension of Y_k increases as new gradients are accepted into the basis for $\mathcal{G}_k$. While iterates linger, the (as yet) unused approximate curvature along directions in $\text{Range}(Y_k)$ continues to be updated.

Lingering on a manifold ends when an RH direction is chosen and x_{k+1} steps "off" $\mathcal{M}(\mathcal{U}_k)$. Once an RH step is taken, the requirement that U_{k+1} be a basis for the subspace of all previously computed RH directions implies that U_{k+1} must be made to include the component of p_k in $\text{Range}(Y_k)$. This update necessitates a corresponding update to R_k. These updates are discussed in sections 3.3–3.4.

3.1. Definition of the basis. At the start of the first iteration, $r_0 = 1$ and Z_0 is just the normalized gradient $g_0/\|g_0\|$, as in Algorithm RH. The initial partition parameter l_0 is zero, which implies that U_0 is void and Y_0 $(= Z_0)$ is $g_0/\|g_0\|$. Since $\mathcal{U}_0$ is empty, it follows that $p_0 \notin \text{Range}(U_0)$, and an RH step is always taken on the first iteration. At the start of the second iteration, if g_1 is rejected, then $Z_1 = Z_0$, Y_1 is void, $U_1 = Z_1$, $r_1 = 1$, and $l_1 = 1$. On the other hand, if g_1 is accepted, then $Z_1 = (z_0 \;\; z_1)$, where $z_0 = g_0/\|g_0\|$ and z_1 is the normalized component of g_1 orthogonal to z_0. In this case $r_1 = 2$ and $l_1 = 1$, which implies that $U_1 = z_0$ and $Y_1 = z_2$. Using the definitions of z_0 and z_1 it can be verified that

$$B_1 = (p_0 \quad g_1) = (z_0 \quad z_1) \begin{pmatrix} \rho_0 & z_0^T g_1 \\ 0 & \rho_1 \end{pmatrix} = Z_1 T_1,$$

where $\rho_0 = \|p_0\|$.

At the start of the kth iteration, the composition of B_k depends on what has occurred in previous iterations. More precisely, we show that B_k is determined by $B_{k-1} = (P_{k-1} \;\; G_{k-1})$ and two decisions made during iteration $k-1$: (i) the choice of p_{k-1} (i.e., whether it defines an RH or a lingering step), and (ii) the result of the orthogonalization procedure (i.e., whether or not g_k is accepted at the end of iteration $k-1$).

Next, we consider the choice of search direction. Suppose p_{k-1} is an RH direction. Given B_{k-1}, the definition of the new basis B_k involves a *two-stage* procedure in which an intermediate basis B'_{k-1} is defined from matrices P'_{k-1} and G'_{k-1}. The matrix P'_{k-1} is defined by appending p_{k-1} to the right of P_{k-1} to give $P'_{k-1} = (P_{k-1} \;\; p_k)$. The RH direction p_{k-1} must, by definition, satisfy $p_{k-1} \in \text{Range}(P_{k-1} \;\; G_{k-1})$, and hence $(P_k \;\; G_{k-1}) = (P_{k-1} \;\; p_{k-1} \;\; G_{k-1})$ will always have dependent columns. To maintain a linearly independent set of basis vectors, it is therefore necessary to define G'_{k-1} as G_{k-1} with one of its columns removed. When a column is removed from G_{k-1}, the matrices Z_{k-1} and R_{k-1} must be updated. The work needed for this is least if the *last* column is deleted from G_{k-1} (see section 3.3). This procedure corresponds to discarding the most recently computed gradient remaining in G_{k-1}, say g_{k-j} $(k \geq j > 0)$. Note that this deletion procedure is always well defined since G_{k-1} cannot be void when p_{k-1} is an RH direction. Now assume that p_{k-1} is a lingering direction. In this case, we define $P'_{k-1} = P_{k-1}$ and $G'_{k-1} = G_{k-1}$.

The second stage in the calculation of B_k is the definition of P_k and G_k from P'_{k-1} and G'_{k-1} after the orthogonalization procedure of iteration $k-1$. Under the assumption that g_k is accepted, we define $P_k = P'_{k-1}$ and $G_k = (G'_{k-1} \;\; g_k)$. If g_k is not accepted, then $P_k = P'_{k-1}$ and $G_k = G'_{k-1}$.

TABLE 3.1
Example of the composition of P_k and G_k.

k	P_k	G_k	p_k	g_{k+1}
0	*void*	(g_0)	RH	accepted
1	(p_0)	(g_1)	RH	rejected
2	$(p_0\ p_1)$	*void*	lingering	accepted
3	$(p_0\ p_1)$	(g_3)	lingering	accepted
4	$(p_0\ p_1)$	$(g_3\ g_4)$	lingering	accepted
5	$(p_0\ p_1)$	$(g_3\ g_4\ g_5)$	lingering	accepted
6	$(p_0\ p_1)$	$(g_3\ g_4\ g_5\ g_6)$	RH	accepted
7	$(p_0\ p_1\ p_6)$	$(g_3\ g_4\ g_5\ g_7)$	RH	rejected
8	$(p_0\ p_1\ p_6\ p_7)$	$(g_3\ g_4\ g_5)$	RH	rejected
9	$(p_0\ p_1\ p_6\ p_7\ p_8)$	$(g_3\ g_4)$	lingering	rejected
10	$(p_0\ p_1\ p_6\ p_7\ p_8)$	$(g_3\ g_4)$	–	–

These updating rules provide the basis for an algorithm in which P_k can grow at a rate that is commensurate with the rate at which curvature is being established on the manifold $\mathcal{M}(\mathcal{U}_k)$. To illustrate how P_k can change from one iteration to the next, consider the composition of P_k and G_k for the first ten iterations for a function f with at least seven variables. The iterations are summarized in Table 3.1. Each row of the table depicts quantities computed during a given iteration. The first column gives the iteration number, the next two columns give the composition of P_k and G_k, the fourth column indicates the type of direction used, and the last column states whether or not g_{k+1} is accepted after the line search.

If G_k has one more column than G_{k-1}, then p_{k-1} must be a lingering direction and g_k must be accepted (as is the case for $k = 3$, 4, 5, and 6). Similarly, if G_k has one less column than G_{k-1}, then p_{k-1} must be an RH direction and g_k must be rejected ($k = 2$, 8, 9). The matrix G_k will have the same number of columns as G_{k-1} if p_{k-1} is a lingering direction and g_k is rejected ($k = 10$), or if p_{k-1} is an RH direction and g_k is accepted ($k = 1$, 7).

The column dimension of G_k is the number of accepted gradients with indices between 0 and k less the number of RH directions with indices between 0 and $k-1$. In our example, only g_3 and g_4 remain in G_{10}, although every other gradient lies in $\text{Range}\begin{pmatrix} P_{10} & G_{10} \end{pmatrix}$.

To simplify the notation for the remainder of this section, the index k is omitted and overbars indicate quantities associated with iteration $k+1$.

3.2. Definition of the search direction. Next we consider the definition of the lingering and RH search directions, and give a method for choosing between them.

If the rows and columns of R are partitioned to match the partition $Z = \begin{pmatrix} U & Y \end{pmatrix}$, we obtain

$$R = \begin{pmatrix} R_U & R_{UY} \\ 0 & R_Y \end{pmatrix}, \tag{3.3}$$

where R_U is an $l \times l$ upper-triangular matrix. In terms of this partition, the intermediate system $R^T d = -v$ of Algorithm RH is equivalent to two smaller systems

$$R_U^T d_U = -v_U \quad \text{and} \quad R_Y^T d_Y = -(R_{UY}^T d_U + v_Y),$$

where v_U, v_Y, d_U, and d_Y denote subvectors of v and d corresponding to the U- and Y-parts of Z. Note that the vector $v_U = U^T g$ is the reduced gradient associated with

the subspace Range(U). The RH direction minimizes the quadratic model $\varphi(p)$ in the r-dimensional subspace Range(Z), which includes Range(Y). The RH direction is denoted by p^r to distinguish it from the lingering direction p^l defined below.

If the new iterate $\bar{x}$ is to lie on $\mathcal{M}(\mathcal{U})$, the search direction must lie in Range(U). The obvious choice for p^l is the unique minimizer of the local quadratic model $\varphi(p) = g^T p + \frac{1}{2} p^T H p$ in Range(U). This minimizer is given by $-U(U^T H U)^{-1} U^T g$, from which it follows that p^l can be computed as $p^l = U R_U^{-1} d_U$.

The choice between p^r and p^l is based on comparing $\varphi(p^r)$ with $\varphi(p^l)$, where the quadratic model $\varphi(p)$ estimates $f(x+p) - f(x)$, the change in the objective. From the definitions of p^r and p^l, we have

$$\varphi(p^r) = -\tfrac{1}{2}\|d\|^2 \quad \text{and} \quad \varphi(p^l) = -\tfrac{1}{2}\|d_U\|^2.$$

These predictions are attained if f is a convex quadratic and an exact line search is used. In this case the gradients are mutually orthogonal (see section 2.3), both v_U and d_U are zero, and the only way to decrease f is to step off $\mathcal{M}(\mathcal{U})$ using p^r.

On the other hand, when minimizing a general nonlinear function with an inexact line search, it is possible that $\|d_U\| \approx \|d\|$, and nearly all of the reduction in the quadratic model is obtained on $\mathcal{M}(\mathcal{U})$. In this event, little is lost by forcing the iterates to remain on $\mathcal{M}(\mathcal{U})$. In addition, lingering can be used to ensure that the reduced gradient v_U is "sufficiently small," and may help to further establish the curvature on $\mathcal{U}$. In this sense, lingering is a way of forcing Broyden's method to perform on a general nonlinear function as it does on a quadratic.

As noted by Fenelon [4, p. 72], it can be inefficient to remain on $\mathcal{M}(\mathcal{U})$ until the reduced gradient v_U is zero. Instead, iterates are allowed to linger until the predicted reduction corresponding to a step moving off of $\mathcal{M}(\mathcal{U})$ is significantly better than the predicted reduction for a step that lingers. In particular, a step off of $\mathcal{M}(\mathcal{U})$ is taken if $\|d_U\|^2 \le \tau \|d\|^2$, where τ is a preassigned constant such that $\frac{1}{2} < \tau < 1$.

The following simple argument shows that if $p = p^r$ is selected when the condition $\|d_U\|^2 \le \tau\|d\|^2$ is satisfied, then the next iterate steps off of $\mathcal{M}(\mathcal{U})$. If the U- and Y-parts of q are denoted by q_U and q_Y, respectively, the partitioned form of $Rq = d$ is given by

$$R_Y q_Y = d_Y \quad \text{and} \quad R_U q_U = d_U - R_{UY} q_Y. \tag{3.4}$$

Written in terms of p_U and p_Y, the search direction satisfies $p = U q_U + Y q_Y$. The inequality $(1-\tau)\|d_U\|^2 \le \tau \|d_Y\|^2$ implies that both d and d_Y are nonzero, and it follows from (3.4) and the nonsingularity of R_Y that q_Y is nonzero. Hence, $Y q_Y$ is also nonzero and $\bar{x} = x + \alpha p^r$ steps off of $\mathcal{M}(\mathcal{U})$.

3.3. Updating *Z*. Let P, G, T, and Z denote matrices associated with the orthogonal factorization (3.2) at the start of an iteration. In section 3.1 it was shown that the basis undergoes two (possibly trivial) changes during an iteration, i.e., $B = (\,P \;\; G\,) \to B' = (\,P' \;\; G'\,) \to \bar{B} = (\,\bar{P} \;\; \bar{G}\,)$.

The first change to Z involves updating the orthogonal factorization $(\,P \;\; G\,) = ZT = (\,U \;\; Y\,)T$ to obtain $(\,P' \;\; G'\,) = Z'T' = (\,U' \;\; Y'\,)T'$, associated with the intermediate basis B'. The update depends on the choice of p. If p is the lingering direction p^l, we have the trivial case $T' = T$, $U' = U$, and $Y' = Y$. If p is the RH direction p^r, then $P' = (\,P \;\; p\,)$ and the resulting effect on U and Y must be calculated.

Introducing p on both sides of the decomposition $\begin{pmatrix} P & G \end{pmatrix} = ZT$ yields

$$\begin{pmatrix} P & p & G \end{pmatrix} = \begin{pmatrix} U & Y \end{pmatrix} \begin{pmatrix} T_U & q_U & T_{UY} \\ 0 & q_Y & T_Y \end{pmatrix},$$

where $q = Z^T p$ and q_U and q_Y denote the components of q corresponding to the U- and Y-parts of Z. The left-hand side can be repartitioned as $(P' \quad G' \quad \underline{g})$, where $P' = \begin{pmatrix} P & p \end{pmatrix}$ and $\underline{g}$ is the most recently accepted gradient remaining in the basis. Let S denote an $r \times r$ orthogonal upper-Hessenberg matrix constructed such that

$$S = \begin{pmatrix} I_l & 0 \\ 0 & S_Y \end{pmatrix} \quad \text{and} \quad Sq = \begin{pmatrix} q_U \\ \|q_Y\| e_1 \end{pmatrix}.$$

It follows that

$$\text{(3.5)} \quad \begin{pmatrix} P' & G' & \underline{g} \end{pmatrix} = \begin{pmatrix} U & Y S_Y^T \end{pmatrix} T_S, \quad \text{where} \quad T_S = \begin{pmatrix} T_U & q_U & T_{UY} \\ 0 & S_Y q_Y & S_Y T_Y \end{pmatrix}.$$

The shape of S implies that the $(r-l) \times (r-l+1)$ matrix $\begin{pmatrix} S_Y q_Y & S_Y T_Y \end{pmatrix}$ is upper-Hessenberg, and the $r \times (r+1)$ matrix T_S is upper-trapezoidal. Deleting the last column from each side of the identity (3.5) gives the required factorization. In particular, $U' = (U \;\; Y S_Y^T e_1)$, $Y' = (Y S_Y^T e_2 \;\; Y S_Y^T e_3 \;\; \cdots \;\; Y S_Y^T e_{r-l})$, $Z' = (U' \;\; Y')$, and $T' = T_S E_r$, where E_r denotes the matrix of first r columns of I_{r+1}.

The matrix S is defined as a product of plane rotations and need not be stored explicitly. One choice of S that uses symmetric Givens matrices instead of plane rotations is given by Daniel et al. [2] in the context of inserting a column into a QR factorization. As S can be generated entirely from q_Y, the matrix T need not be stored.

If $l < r-1$, then the modification of U and Y requires approximately $3(r-l-1)n$ flops (see Daniel et al. [2]). If $l = r-1$, then no work is required since the columns of $Z = \begin{pmatrix} U & Y \end{pmatrix}$ are already an orthonormal basis for $\begin{pmatrix} P & p \end{pmatrix}$. (The argument is similar to that given in Lemma 2.2, although here q_Y is nonzero according to the reasoning given at the end of section 3.2.)

The second stage in updating Z is to compensate for the change from B' to the final basis $\bar{B}$. After the line search, if the new gradient $\bar{g}$ is rejected, then we have the trivial case $\bar{T} = T'$, $\bar{U} = U'$, and $\bar{Y} = Y'$. If $\bar{g}$ is accepted, then $\bar{\rho}\bar{z} = \bar{g} - Z'Z'^T\bar{g}$ defines the normalized component of $\bar{g}$ outside Range(Z') (see section 2.1). In this case, the identity

$$\begin{pmatrix} P' & G' & \bar{g} \end{pmatrix} = \begin{pmatrix} Z' & \bar{z} \end{pmatrix} \begin{pmatrix} T' & Z'^T\bar{g} \\ 0 & \bar{\rho} \end{pmatrix}$$

implies that $\bar{P} = P'$, $\bar{G} = \begin{pmatrix} G' & \bar{g} \end{pmatrix}$, $\bar{U} = U'$, $\bar{Y} = (Y' \;\; \bar{z})$, and $\bar{T}$ is T' augmented by the column $\bar{Z}^T\bar{g}$. In both cases, we define $\bar{Z} = (\bar{U} \;\; \bar{Y})$.

3.4. Updating *R*. When Z is updated to include the new RH direction, the new reduced Hessian is $Z'^T H Z' = S Z^T H Z S^T = S R^T R S^T$, where H is given by (2.10). The $(2,2)$ block of RS^T is $R_Y S_Y^T$, which can be restored to upper-triangular form using a suitable sequence of plane rotations S' applied on the left of R. This

results in $R_Y S_Y^T$ being premultiplied by an orthogonal matrix S'_Y such that $S'_Y R_Y S_Y^T$ is upper-triangular. The Cholesky factor of $Z'^T H Z'$ is then

$$R' = S' R S^T = \begin{pmatrix} R_U & R_{UY} S_Y^T \\ 0 & S'_Y R_Y S_Y^T \end{pmatrix} = \begin{pmatrix} R'_{U'} & R'_{U'Y'} \\ 0 & R'_{Y'} \end{pmatrix},$$

where $R'_{U'}$ and $R'_{Y'}$ are upper-triangular matrices of order $l+1$ and $r-l-1$. For more details, see Leonard [16, p. 40]. The calculation of R' simplifies considerably if the BFGS update is used (see section 3.7).

It remains to update R' to reflect the second stage of the basis change: $B' = (\, P' \quad G' \,) \rightarrow \bar{B} = (\, \bar{P} \quad \bar{G} \,)$, which corresponds to the orthogonalization of the new gradient. If R'' denotes the updated factor, then R'' is obtained from R' by adding a scaled unit row and column, as in (2.8).

3.5. Updating related quantities. After the new gradient has been orthogonalized, the vectors $u'' = \bar{Z}^T \bar{g}$, $v'' = \bar{Z}^T g$, and $q'' = \bar{Z}^T p$ are used to define the quasi-Newton update $\bar{R} = \textbf{\textit{Broyden}}(R'', s, y)$ with $s = \alpha q''$ and $y = u'' - v''$. The vector u'' is computed as a by-product of the orthogonalization, as in Algorithm RH. The vectors v'' and q'' can be computed from v and q using intermediate vectors $v' = Z'^T g$ and $q' = Z'^T p$ in conjunction with the two-stage update to B. If p is a lingering direction, then $v' = v$ and $q' = q$. Otherwise, the definition of Z' implies that

$$v' = Z'^T g = S Z^T g = S v = \begin{pmatrix} v_U \\ S_Y v_Y \end{pmatrix},$$

which can be computed efficiently by applying the plane rotations of S as they are generated. Similarly, the U'- and Y'-portions of q' are $q'_{U'} = (q_U, \|q_Y\|)^T$ and $q'_{Y'} = 0$, since $S_Y q_Y = \|q_Y\| e_1$. These expressions provide a cheaper alternative to computing the RH search direction as $p = U q_U + Y q_Y$. With this alternative, U is modified as soon as q_U and q_Y are known, and p is computed from the expression $p = U' q'_{U'}$.

Once v' and q' are known, v'' and q'' are found from v' and q' during the orthogonalization procedure as in Algorithm RH.

3.6. A reduced-Hessian method with lingering. We summarize the results of this section by describing a complete reduced-Hessian method with lingering. As in Algorithm RH of section 2.2, certain calculations are represented schematically as functions with input and output arguments. The first stage of the basis update can be viewed as swapping the new RH direction with the most recently accepted gradient remaining in B_k. Accordingly, the modification of Z_k (and hence U_k and Y_k) and related quantities is represented by $(Z'_k, R'_k, q'_k, v'_k) = \textbf{\textit{swap}}(Z_k, R_k, q_k, v_k)$.

ALGORITHM 3.1. REDUCED-HESSIAN METHOD WITH LINGERING (RHL).

Choose x_0 and σ ($\sigma > 0$);
$k = 0$; $r_0 = 1$; $l_0 = 0$; $g_0 = \nabla f(x_0)$;
$Z_0 = g_0/\|g_0\|$; $R_0 = \sigma^{1/2}$; $v_0 = \|g_0\|$;
while not converged **do**
  Partition R_k as R_U, R_Y, and R_{UY}; Partition v_k as v_U and v_Y;
  Solve $R_U^T d_U = -v_U$; $R_Y^T d_Y = -(R_{UY}^T d_U + v_Y)$;
  if $\|d_U\|^2 > \tau \|d\|^2$ **then**
    Solve $R_U q_U = d_U$; $q_Y = 0$;

$Z'_k = Z_k$; $R'_k = R_k$; $q'_k = q_k$; $v'_k = v_k$;
$l'_k = l_k$;
else
Solve $R_Y q_Y = d_Y$; $R_U q_U = d_U - R_{UY} q_Y$;
$(Z'_k, R'_k, q'_k, v'_k) = \boldsymbol{swap}(Z_k, R_k, q_k, v_k)$;
$l'_k = l_k + 1$;
end if
$p_k = U'_k q'_{U'}$; $l_{k+1} = l'_k$;
Find α_k satisfying the Wolfe conditions (2.4);
$x_{k+1} = x_k + \alpha_k p_k$; $g_{k+1} = \nabla f(x_k + \alpha_k p_k)$; $u_k = Z_k^T g_{k+1}$;
$(Z_{k+1}, r_{k+1}, R''_k, u''_k, v''_k, q''_k) = \boldsymbol{expand}(Z'_k, r_k, R'_k, u_k, v'_k, q'_k, g_{k+1}, \sigma)$;
$s_k = \alpha_k q''_k$; $y_k = u''_k - v''_k$; $R_{k+1} = \boldsymbol{Broyden}(R''_k, s_k, y_k)$;
$v_{k+1} = u''_k$; $k \leftarrow k + 1$;
end do

As in Algorithm RH, no new gradients are accepted once r_k reaches n. If the test $\|d_U\|^2 \leq \tau \|d\|^2$ is satisfied every iteration, Algorithm RHL generates the same sequence of iterates as Algorithm RH. In this case, every iteration starts with $l_k = r_k$ or $l_k = r_k - 1$. If $l_k = r_k$, the previous gradient g_k was rejected and both algorithms compute a lingering direction. Otherwise, if $l_k = r_k - 1$, then g_k was accepted and Y_k must have just one column. Once the RH direction is computed, the swap procedure amounts to moving the partition of Z_k so that the Y-part becomes void. It follows that if only RH directions are chosen, the partition of Z_k is used only to decide if p_k is an RH or lingering direction.

3.7. The BFGS update. If the BFGS update is used, the block structure (3.3) of R simplifies to the extent that R_Y is always $\sigma^{1/2} I_{r-l}$. This can be shown using two results. The first describes the effect of the BFGS update on R when $s \in \text{Range}(U)$.

LEMMA 3.1. *Let R denote an $r \times r$ nonsingular upper-triangular matrix partitioned as in* (3.3). *Let y and s be r-vectors such that $y^T s > 0$. If the Y-components of s are zero, then the update $\bar{R} = \boldsymbol{BFGS}(R, s, y)$ does not alter the* $(2,2)$ *block of R (i.e., $\bar{R}_Y = R_Y$). Moreover, $\bar{R}_U$ and $\bar{R}_{UY}$ are independent of R_Y.*

Proof. The result follows from the definition (2.9) of the rank-one BFGS update to R (see Leonard [16, pp. 13–15] for the first part). □

The next lemma considers the cumulative effect of Algorithm RHL on the block structure of R.

LEMMA 3.2. *Assume that Algorithm RHRL is used with the BFGS update, and that Z is partitioned as $Z = \begin{pmatrix} U & Y \end{pmatrix}$. Then there exist orthogonal matrices S and S' for the basis update such that, at the start of every iteration, R has the form*

$$\begin{pmatrix} R_U & R_{UY} \\ 0 & R_Y \end{pmatrix} \quad \text{with} \quad R_Y = \sigma^{1/2} I_{r-l}. \tag{3.6}$$

Proof. The proof is by induction. The result holds at the start of the first iteration since $r_0 = 1$, $l_0 = 0$, and $R_Y = \sigma^{1/2}$. Assume that the result holds at the start of iteration k.

If the partition parameter is increased, the columns of Y are modified and the $(2,2)$ block of R' satisfies $R'_Y = S'_Y R_Y S_Y^T = \sigma S'_Y S_Y^T$. If $S' = S$, then $R'_Y = \sigma I_{r-l}$. If the partition parameter does not change, then $R' = R$ and $R'_Y = \sigma I_{r-l}$ trivially.

The repartition resulting from the change in l gives $\sigma^{1/2} I_{r-\bar{l}}$ in the $(2,2)$ block, and it follows that $R'_{Y'}$ has the required form prior to the line search. Note that $R'_{Y'}$ is void if either R_Y is void (i.e., $l = r$) or l was increased to r (giving $\bar{l} = r$).

The expansion procedure may add a scaled unit row and column to R'. In either case, R'' can be partitioned to match $\bar{U}$ and $\bar{Y}$ as

$$R'' = \begin{pmatrix} R''_{\bar{U}} & R''_{\bar{U}\bar{Y}} \\ 0 & R''_{\bar{Y}} \end{pmatrix}.$$

It follows that $R''_{\bar{Y}} = \sigma^{1/2} I_{\bar{r}-\bar{l}}$.

Whatever the choice of search direction, q'' is of the form $q'' = (q''_{\bar{U}}, 0)^T$, where $q''_{\bar{U}}$ is an $\bar{l}$-vector. Thus, R'' and s satisfy the conditions of Lemma 3.1, and $\bar{R} = \boldsymbol{BFGS}(R'', s, y)$ has the required structure. □

If, in the BFGS case, instead of defining $S' = S$, we update R_Y according to the procedure of section 3.3, then the updated matrix will be of the form $R_Y = \sigma^{1/2} \tilde{I}_{r-l}$, where $\tilde{I}_{r-l}$ is a diagonal matrix of plus or minus ones. The purpose of Lemma 3.2 is to show that it is *unnecessary* to apply S^T and S' to R_Y when RHL is used with the BFGS update. Instead, S_Y^T need only be applied to R_{UY}, at a cost of $3(r-l-1)l$ flops.

3.8. Operation count for RHL with the BFGS update. The number of operations for an iteration of the BFGS version of Algorithm RHL will depend on the type of search direction selected. If a lingering direction is used, the vector $R^T R q$ will be different from $-v$, and the vector v cannot be substituted for the matrix-vector product in (2.9). However, in this case we have

$$R^T R q = \begin{pmatrix} R_U^T R_U q_U \\ R_{UY}^T R_U q_U \end{pmatrix} = \begin{pmatrix} -v_U \\ R_{UY}^T R_U q_U \end{pmatrix},$$

which requires only $R_{UY}^T R_U q_U$ to be computed explicitly.

Whichever search direction is used, the vector s has $\bar{r} - \bar{l}$ trailing zeros (see (2.9)), and the cost of applying the BFGS update drops to $6\bar{r}\bar{l} - 3\bar{l}^2$ flops. It follows that iterations involving a lingering direction require $(2r+l+1)n + \frac{1}{2}r^2 + 7rl - \frac{7}{2}l^2 + \mathcal{O}(r) + \mathcal{O}(l)$ flops. If $l = r$, the work is commensurate with that of Algorithm RH. If an RH step is taken, an additional n flops are required because p is a linear combination of $l+1$ n-vectors instead of l n-vectors. In this case, $3(r-l-1)(l+n)$ flops are required to update Z and R using the method of sections 3.3–3.4.

4. Modifying approximate curvature. The choice of H_0 can greatly influence the practical performance of quasi-Newton methods. The usual choice $H_0 = \sigma I$ ($\sigma > 0$) can result in many iterations and function evaluations—especially if $\nabla^2 f(x^*)$ is ill-conditioned (see, e.g., Powell [25] and Siegel [28]). This is sometimes associated with "stalling" of the iterates, a phenomenon that can greatly increase the overall cpu time for solution (or termination).

To date, the principal modifications of conventional quasi-Newton methods have involved *scaling* all or part of the approximate Hessian. Several scaling methods have appeared in the literature. In the *self-scaling variable metric (SSVM)* method of Oren and Luenberger [24], H_k is multiplied by a positive scalar prior to application of the Broyden update. The *conjugate-direction scaling* method of Siegel [28] scales columns of a certain conjugate-direction factorization of H_k^{-1}. This scheme, which

is a refinement of a method of Powell [25], has been shown to be globally and q-superlinearly convergent. In what follows, Siegel's method will be referred to as Algorithm CDS. Finally, Lalee and Nocedal [15] have proposed an algorithm that scales columns of a lower-Hessenberg factor of H_k. This method will be referred to as Algorithm ACS, which stands for *automatic column scaling*.

Here, scaling takes the form of resetting certain diagonal elements of the Cholesky factor of the reduced-Hessian. The structure of the transformed Hessian $Q_k^T H_k Q_k$ (2.5) reveals the influence of H_0 on the approximate Hessian. For example, the initial Hessian scale factor σ represents the approximate curvature along all unit directions in $\mathcal{G}_k^{\perp}$ (see Lemma 2.1). Inefficiencies resulting from poor choices of H_0 may be alleviated by maintaining a current estimate σ_k of the approximate curvature in $\mathcal{G}_k^{\perp}$. At the end of each iteration, the new estimate σ_{k+1} replaces σ_k in the transformed Hessian wherever this can be done without endangering its positive definiteness. This replacement has the effect of *reinitializing* approximate curvature along all directions in $\mathcal{G}_k^{\perp}$, and along certain directions in $\mathcal{G}_k$. In the next section, an algorithm of this type is introduced as a generalization of Algorithm RHL.

4.1. Reinitialization combined with lingering. In this section we extend the BFGS version of Algorithm RHL so that approximate curvature is modified in a subspace of dimension $n - \bar{l}$ immediately following the BFGS update. We choose to emphasize the BFGS method because the diagonal structure $\bar{R}_{\bar{Y}} = \sigma^{1/2} I_{\bar{r}-\bar{l}}$ of the $(2,2)$ block of the BFGS Cholesky factor reveals the main influence of H_0 on the approximate Hessian. In this case, the initial approximate curvature along unit directions in Range$(\bar{Y})$ is explicit and easily reinitialized. The approximate curvature along directions in Range$(\bar{U})$ is considered to be sufficiently established (in the sense of Lemma 2.4) and hence the corresponding part of the reduced Hessian is unaltered.

Reinitialization is not hard to achieve in comparison to some scaling procedures previously proposed. Reinitialization simply involves replacing the factor

$$R''' = \begin{pmatrix} R'''_{\bar{U}} & R'''_{\bar{U}\bar{Y}} \\ 0 & \sigma^{1/2} I_{\bar{r}-\bar{l}} \end{pmatrix} \quad \text{by} \quad \bar{R} = \begin{pmatrix} R'''_{\bar{U}} & R'''_{\bar{U}\bar{Y}} \\ 0 & \bar{\sigma}^{1/2} I_{\bar{r}-\bar{l}} \end{pmatrix},$$

where the matrix R''' is the final factor obtained in an iteration of Algorithm RHL. The corresponding effect on the $(2,2)$ block of the reduced Hessian is to replace the term $\sigma I_{\bar{r}-\bar{l}}$ by $\bar{\sigma} I_{\bar{r}-\bar{l}}$.

This reinitialization exploits the special structure of R''' resulting from the lingering scheme. The resulting method may be compared to Liu and Nocedal's limited-memory L-BFGS method [17]. In this case, the BFGS *inverse* Hessian is defined as the last of a sequence of auxiliary inverse Hessians generated implicitly from σI and a set of vector pairs (δ_k, γ_k) (see (2.3)). This form allows σI to be reinitialized at every iteration (in which case, every auxiliary inverse Hessian is changed). The fact that the rank-two terms are not summed explicitly is crucial. If the inverse Hessian were to be stored elementwise, then any reinitialization that adds a (possibly negative-definite) diagonal $(\bar{\sigma} - \sigma)I$ would leave all the auxiliary approximations unchanged except the first, and thereby define a potentially indefinite approximation. In the reduced-Hessian formulation, it is possible to maintain an elementwise approximation *and* reinitialize unestablished curvature without risk of indefiniteness. The diagonal form of $\bar{R}_{\bar{Y}}$ means that σ occurs as an explicit modifiable term in the expression for the curvature along directions in Range$(\bar{Y})$. This term can be safely reset to any positive number $\bar{\sigma}$.

It remains to define an appropriate value for $\bar{\sigma}$. We consider four alternatives that have been effective in practice. The first two are the simple choices:

$$\sigma_{k+1}^{R0} = 1 \quad \text{and} \quad \sigma_{k+1}^{R1} = \frac{y_0^T y_0}{y_0^T s_0} \tag{4.1}$$

(see Shanno and Phua [26] for a discussion of σ_{k+1}^{R1}). The third alternative is related to the scaling parameters used in Algorithm CDS (see Siegel [28]). It is defined in terms of a scalar γ_i and satisfies

$$\sigma_{k+1}^{R2} = \min_{0 \le i \le k} \{\gamma_i\}, \quad \text{where} \quad \gamma_i = \frac{y_i^T s_i}{\|s_i\|^2}. \tag{4.2}$$

The fourth alternative is the inverse of the value suggested by Liu and Nocedal [17] for use in the limited-memory BFGS method (see Nocedal [23]). For this option, we define

$$\sigma_{k+1}^{R3} = \frac{y_k^T y_k}{y_k^T s_k}. \tag{4.3}$$

Reinitialization is represented schematically as $\bar{R} = \textbf{\textit{reinitialize}}(R''', \bar{\sigma})$ in the algorithm below.

ALGORITHM 4.1. REDUCED-HESSIAN METHOD WITH REINITIALIZATION AND LINGERING (RHRL).

Choose x_0 and σ_0 ($\sigma_0 > 0$);
$k = 0$; $r_0 = 1$; $l_0 = 0$; $g_0 = \nabla f(x_0)$;
$Z_0 = g_0/\|g_0\|$; $R_0 = \sigma_0^{1/2}$; $v_0 = \|g_0\|$;
while not converged **do**
  Partition R_k as R_U, R_Y and R_{UY}; Partition v_k as v_U and v_Y;
  Solve $R_U^T d_U = -v_U$; $R_Y^T d_Y = -(R_{UY}^T d_U + v_Y)$;
  if $\|d_U\|^2 > \tau \|d\|^2$ **then**
    Solve $R_U q_U = d_U$; $q_Y = 0$;
    $Z_k' = Z_k$; $R_k' = R_k$; $q_k' = q_k$; $v_k' = v_k$;
    $l_k' = l_k$;
  else
    Solve $R_Y q_Y = d_Y$; $R_U q_U = d_U - R_{UY} q_Y$;
    $(Z_k', R_k', q_k', v_k') = \textbf{\textit{swap}}(Z_k, R_k, q_k, v_k)$;
    $l_k' = l_k + 1$;
  end if
  $p_k = U_k' q_{U'}'$; $l_{k+1} = l_k'$;
  Find α_k satisfying the Wolfe conditions (2.4);
  $x_{k+1} = x_k + \alpha_k p_k$; $g_{k+1} = \nabla f(x_k + \alpha_k p_k)$; $u_k = Z_k^T g_{k+1}$;
  $(Z_{k+1}, r_{k+1}, R_k'', u_k'', v_k'', q_k'') = \textbf{\textit{expand}}(Z_k', r_k, R_k', u_k, v_k', q_k', g_{k+1}, \sigma_k)$;
  $s_k = \alpha_k q_k''$; $y_k = u_k'' - v_k''$; $R_k''' = \textbf{\textit{BFGS}}(R_k'', s_k, y_k)$;
  Compute σ_{k+1}; $R_{k+1} = \textbf{\textit{reinitialize}}(R_k''', \sigma_{k+1})$;
  $v_{k+1} = u_k''$; $k \leftarrow k + 1$;
end do

Other than the addition of the ***reinitialize*** procedure, Algorithm RHRL differs from RHL only in the specific use of the BFGS update.

Reinitialization can be applied directly to Algorithm RH by redefining σ *before* the ***expand*** and ***Broyden*** procedures. When the BFGS update is used and R_k expands, the last diagonal of R_k'' is unaltered and is independent of the remainder of $\bar{R}$ (see section 3.7). In this case, the last diagonal can be redefined either before or after the update. This option is also available for RHRL, but reinitialization is done after the BFGS update to simplify the proof of Theorem 4.3. (The trailing columns of the conjugate-direction matrix are scaled *after* the BFGS update in Algorithm CDS [28].)

4.2. Algorithm RHRL applied to a quadratic. Consider the strictly convex quadratic function $f(x) = c - b^T x + \frac{1}{2}x^T A x$ of (2.11). The next theorem extends Fenelon's quadratic termination results for Algorithm RH to Algorithm RHRL (see section 2.3). In the statement of the theorem, r_{ij} denotes the (i,j)th component of R_k. For details of the proof, see Leonard [16, pp. 58–61].

THEOREM 4.1. *Consider Algorithm RHRL applied with an exact line search and $\sigma_0 = 1$ to minimize the quadratic $f(x)$ of* (2.11). *Then, at the start of iteration k, the rank of R_k is $r_k = k+1$, the partition parameter is $l_k = k$, and Z_k satisfies $Z_k = (\,U_k \;\; Y_k\,)$, where the columns of U_k are the normalized gradients $\{g_i/\|g_i\|\}$, $1 \le i \le k-1$, and $Y_k = g_k/\|g_k\|$. Moreover, the upper-triangular matrix R_k is upper bidiagonal with $R^k_{UY} = -\|g_k\|/(y^T_{k-1}s_{k-1})e_k$ and $R^k_Y = \sigma_k^{1/2}$. The nonzero components of R_k in R^k_U satisfy $r_{ii} = \|g_{i-1}\|/(y^T_{i-1}s_{i-1})^{1/2}$ and $r_{i,i+1} = -\|g_i\|/(y^T_{i-1}s_{i-1})^{1/2}$ for $1 \le i \le k$. Furthermore, the search directions satisfy*

$$p_0 = -g_0; \qquad p_k = -\frac{1}{\sigma_k}g_k + \beta_{k-1}p_{k-1}, \quad \beta_{k-1} = \frac{\sigma_{k-1}}{\sigma_k}\frac{\|g_k\|^2}{\|g_{k-1}\|^2}, \quad k \ge 1. \qquad \square$$

COROLLARY 4.2. *If Algorithm RHRL is applied with an exact line search to minimize the quadratic $f(x)$ of* (2.11), *and $\sigma_0 = 1$, then the minimizer will be found in at most n iterations.*

Proof. We show by induction that the search directions are parallel to the conjugate-gradient directions $\{d_k\}$. Specifically, $\sigma_k p_k = d_k$ for all k. This is true for $k = 0$ since $\sigma_0 p_0 = -g_0 = d_0$. Assume that $\sigma_{k-1}p_{k-1} = d_{k-1}$. Using Theorem 4.1 and the inductive hypothesis, we find

$$\sigma_k p_k = -g_k + \sigma_{k-1}\frac{\|g_k\|^2}{\|g_{k-1}\|^2}p_{k-1} = -g_k + \frac{\|g_k\|^2}{\|g_{k-1}\|^2}d_{k-1} = d_k,$$

which completes the induction. Since the conjugate-gradient method has quadratic termination under the assumptions of the theorem, Algorithm RHRL must also have this property. $\square$

4.3. An equivalence with conjugate-direction scaling. The next theorem, proved by Leonard [16, pp. 62–77], states that under certain conditions, Algorithm RHRL generates the same iterates as the CDS algorithm of Siegel [28].

THEOREM 4.3. *Suppose that Algorithm* RHRL *and Algorithm* CDS *are used to find a local minimizer of a twice-continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. If both algorithms start from the same point and use the same line search, and if* RHRL *uses $\sigma_0 = 1$, $\sigma_k = \sigma_k^{R2}$, $\tau = \frac{10}{11}$, then the algorithms generate the same sequence of iterates.* $\square$

Despite this equivalence, we emphasize that RHRL and CDS are *not the same method*. First, the stated equivalence concerns CDS and one instance of RHRL, so

RHRL may be considered as a generalization of CDS. Second, the dimensions of the matrices required and the computation times differ substantially for the two methods. CDS has a 33% advantage with respect to storage, since RHRL requires $\frac{3}{2}n^2$ elements for Z_k and R_k, assuming that they grow to full size. However, RHRL requires substantially lower cpu times in practice—principally because of the more efficient calculation of p_k when k is small relative to n (see section 6.5).

The last result of this section gives convergence properties of Algorithm RHRL when applied to strictly convex functions.

COROLLARY 4.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ denote a strictly convex, twice-continuously differentiable function. Moreover, assume that $\nabla^2 f(x)$ is Lipschitz continuous with $\|\nabla^2 f(x)^{-1}\|$ bounded above for all x in the level set of $f(x_0)$. If Algorithm RHRL with $\sigma_0 = 1$, $\sigma_k = \sigma_k^{R2}$, $\tau = \frac{10}{11}$, and a Wolfe line search is used to minimize f, then convergence is global and q-superlinear.*

Proof. Since the conjugate-direction scaling algorithm has these convergence properties (see Siegel [28]), the proof is immediate from Theorem 4.3. ∎

5. Implementation details. In this section, we discuss the implementation of Algorithm RHRL. Numerical results are given in section 6.

5.1. Reorthogonalization. In exact arithmetic, a gradient is accepted into the basis $B_k' = \begin{pmatrix} P_k' & G_k' \end{pmatrix}$ if $\rho_{k+1} > 0$, where ρ_{k+1} is the two-norm of $(I - Z_k' Z_k'^T) g_{k+1}$. This ensures that the basis vectors are linearly independent, and that the implicitly defined matrix T_k' (3.2) is nonsingular. When ρ_{k+1} is computed in finite precision, gradients with small (but nonzero) ρ_{k+1} are rejected to discourage $\{T_k\}$ from becoming too ill-conditioned. In practice, an accepted gradient must satisfy $\rho_{k+1} \geq \epsilon \|g_{k+1}\|$, where ϵ is a preassigned positive constant. In the numerical results presented in section 6, ϵ was set at 10^{-4}.

Even when ϵ is large relative to the machine precision, Gram–Schmidt orthogonalization is unstable (see Golub and Van Loan [13, p. 218]). Two of the best known algorithms for stabilizing the process are modified Gram–Schmidt and Gram–Schmidt with reorthogonalization (see Golub and Van Loan [13, p. 218] and Daniel et al. [2]). We have used Gram–Schmidt with reorthogonalization in our implementation. Each reorthogonalization requires an additional $2nr_k$ flops.

5.2. The line search, BFGS update, and termination criterion. The line search for the reduced-Hessian methods is a slightly modified version of that used in the package NPSOL [10]. It is designed to ensure that α_k satisfies the strong Wolfe conditions:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + \mu \alpha_k g_k^T p_k \quad \text{and} \quad |g_{k+1}^T p_k| \leq \eta |g_k^T p_k|$$

with $0 \leq \mu \leq \eta < 1$ and $\mu < \frac{1}{2}$. (For more details concerning algorithms designed to meet these criteria, see, e.g., Gill, Murray, and Wright [11], Fletcher [5], and Moré and Thuente [20].) The step length parameters are $\mu = 10^{-4}$ and $\eta = 0.9$. The line search is based on using a safeguarded polynomial interpolation to find an approximate minimizer of the univariate function $\psi(\alpha) = f(x_k + \alpha p_k) - f(x_k) - \mu \alpha g_k^T p_k$ (see Moré and Sorensen [19]). The step α_k is the first member of a minimizing sequence $\{\alpha_k^i\}$ satisfying the Wolfe conditions. The sequence is always started with $\alpha_k^0 = 1$.

If α_k satisfies the Wolfe conditions, it holds that $y_k^T s_k \geq -(1-\eta)\alpha_k g_k^T p_k > 0$, and hence the BFGS update can be applied without difficulty. On very difficult problems, however, the combination of a poor search direction and a rounding error in f may prevent the line search from satisfying the line search conditions within 20 function

TABLE 6.1
Comparison of RHRL with four reinitialization values on 64 *CUTE problems.*

Option	Itn	Fcn	Cpu
R0	26553	45476	22:26
R1	34815	41327	21:50
R2	25808	39856	20:56
R3	23356	30684	18:01

evaluations. In this case, the search terminates with α_k corresponding to the best value of f found so far. If this α_k defines a strict decrease in f, the minimization continues and the BFGS update is skipped unless $y_k^T s_k \geq \epsilon_M \alpha_k |g_k^T p_k|$, where ϵ_M is the machine precision. If a strict decrease in f is not obtained after 20 function evaluations, then the algorithm is terminated (no restarts are allowed).

Every run was terminated when $\|g_k\| < 10^{-6}$ or $\|g_k\| < \epsilon_M^{0.8}(1 + |f(x_k)|)$. Our intent is to compare methods *when they succeed*, and identify the cases where methods fail.

6. Numerical results. The methods are implemented in double precision Fortran 77 on an SGI O2 with R5000 processor and 64MB of RAM. The test problems are taken from the CUTE collection (see Bongartz et al. [1]).

The test set was constructed using the CUTE interactive `select` tool, which allows identification of groups of problems with certain features:

```
Objective function type         : *
Constraints type                : U
Regularity                      : R
Degree of available derivatives : *
Problem interest                : *
Explicit internal variables     : *
Number of variables             : v
Number of constraints           : 0.
```

Of the 73 problems selected with this specification, *indef* was omitted from the trials because the iterates became unbounded for all the methods. For the remaining problems, the smallest allowable value of n satisfying $n \geq 300$ was chosen, with the following exceptions: Smaller values of n were used for *penalty*3, *mancino*, and *sensors* because they otherwise took too much memory to "decode" using the SIF decoder (compiled with the option "tobig"); a smaller n was used for *penalty*2 because the initial steepest-descent direction for $n = 300$ was unusable by the optimizers; $n = 50$ was used for *chnrosnb* and *errinros* since this was the largest value admitted; and the value $n = 31$ was used for *watson* for the same reason.

Four more problems were identified using the `select` tool with input:

```
Number of variables             : in [    50,   300 ].
```

This resulted in problems *tointgor*, *tointpsp*, *tointqor*, and *hydc*20*ls* being added to the test set. All of these problems have 50 variables except *hydc*20*ls*, which has 99 variables.

We begin our discussion by identifying the "best" implementation of the various

TABLE 6.2
Final nonoptimal gradients for RHRL reinitialization schemes on 5 CUTE problems.

Problem	Reinitialization option			
	R0	R1	R2	R3
bdqrtic	–	1.0E-4	–	1.3E-5
cragglvy	–	9.5E-6	–	–
engval1	2.0E-6	3.0E-6	–	–
fletcbv3	3.8E-1	1.0E-1	–	–
vardim	–	2.1E-5	2.1E-5	2.1E-5

TABLE 6.3
RHRL vs. RHR on 62 *CUTE problems.*

Method	Itn	Fcn	Cpu
RHRL (R3)	19453	20949	16:35
RHR (R0)	25898	43676	22:19
RHR (R1)	31609	35722	19:30
RHR (R2)	25575	35994	21:00
RHR (R3)	25445	27411	18:30

reduced-Hessian methods presented earlier. There follows a numerical comparison between this method and several leading optimization codes, including NPSOL [10], the CDS method [28], and the ACS method [15].

6.1. The benefits of reinitializing curvature. First, we compare an implementation of RHRL using four alternative values of σ_{k+1} (see (4.1)–(4.3)), labeled R0–R3. Table 6.1 gives the total number function evaluations and total cpu time (in minutes and seconds) required for a subset of 64 of the 76 problems. The subset contains the 64 problems for which RHRL succeeded with every choice of σ_{k+1}.

The results clearly indicate that *some form* of approximate curvature reinitialization is beneficial in terms of the overall number of function evaluations. This point is reinforced when RHRL is compared with NPSOL, which has no provision for altering the initial approximate curvature. However, on the CUTE problems, the decrease in function evaluations does not necessarily translate into a large advantage in terms of cpu time. The reason for this is that on the problems where a large difference in function evaluations occurs, the required cpu time is small. For example, on the problem *extrosnb*, the function evaluations/cpu seconds required using R0–R3 are, respectively, 5398/39.6, 4914/20.6, 6764/27.1, and 3418/14.6. Although R3 (i.e., RHRL implemented with reinitialization option R3) offers a large advantage in terms of function evaluations, it gains little advantage in cpu time relative to the *overall* cpu time required for all 64 problems. This is partly because the CUTE problems tend to have objective functions that are cheap to evaluate. (On problem *extrosnb*, RHRL with R0 takes longer than R2 because the final r_k is roughly twice the R2 value. With R0, r_k reaches 67 at iteration 81 and remains at that value until convergence at iteration 3862. With R2, however, r_k reaches only 17 by iteration 81 and is never greater than 35, converging after 4976 iterations.)

None of R0–R3 succeed on problems *arglinb*, *arglinc*, *freuroth*, *hydc*20*ls*, *mancino*, *nonmsqrt*, and *penalty*3. The problems for which at least *one* of R0–R3 fail are *bdqrtic*, *cragglvy*, *engval1*, *fletcbv*3, and *vardim*. Table 6.2 shows which of R0–R3 failed on these five problems by giving the corresponding values for $\|g_k\|$ at the final iterate. It

TABLE 6.4
Final nonoptimal gradients for RHRL and RHR on 7 CUTE problems.

Problem	RHRL (R3)	RHR (R0)	RHR (R1)	RHR (R2)	RHR (R3)
bdqrtic	1.3E-5	–	8.3E-5	–	–
cragglvy	–	–	1.2E-5	–	–
*engval*1	–	–	1.9E-6	–	–
*fletcbv*3	–	3.3E-01	1.4E-1	1.3E-1	6.7E-2
fletchbv	–	–	6.4E+3	–	1.4E+6
*penalty*2	–	1.1E+11	–	–	–
vardim	2.1E-5	–	2.1E-5	2.1E-5	2.1E-5

TABLE 6.5
RHRL vs. NPSOL on 64 CUTE problems.

Method	Itn	Fcn	Cpu
RHRL (R3)	22362	27458	17:05
NPSOL	29204	49420	23:55

should be noted that R2 has no real advantage over R3 in this table because R3 nearly meets the termination criteria on *bdqrtic* (the final objective value is 1.20×10^{-3} for both methods) and because 74 function evaluations are required by R2, compared to 53 for R3. The cpu seconds required by R2 and R3 on *bdqrtic* are 0.38 and 0.28.

6.2. The benefits of lingering. Now we illustrate the benefits of lingering by comparing RHRL with an algorithm, designated RHR, that reinitializes the curvature when a gradient is accepted, but does not linger. Five algorithms were tested: RHR with all four resetting options R0–R3, and RHRL with option R3. The termination criteria were satisfied on 62 of the 76 problems. Table 6.3 gives the total number of iterations, function evaluations, and cpu time required. All five algorithms failed on problems *arglinb*, *arglinc*, *freuroth*, *hydc*20*ls*, *mancino*, *nonmsqrt* and *penalty*3. This leaves seven other problems on which at least one of the five methods failed. The two-norms of the final nonoptimal gradients for these problems are given in Table 6.4.

6.3. RHRL compared with NPSOL. Here we make a numerical comparison between RHRL and the general-purpose constrained solver NPSOL (see Gill et al. [10]). NPSOL uses a Cholesky factor of the approximate Hessian. The code requires approximately $n^2 + \mathcal{O}(n)$ storage locations for unconstrained optimization. The flop count for the method is $4n^2 + \mathcal{O}(n)$ per iteration, with approximately $3n^2$ operations being required for the BFGS update to the Cholesky factor.

In our comparison, both methods meet the termination criteria on 64 of the 76 problems. Table 6.5 gives the total number of iterations, function evaluations and cpu time for RHRL with R3 and for NPSOL. Both methods failed on problems *arglinb*, *arglinc*, *hydc*20*ls*, *mancino*, *nonmsqrt*, and *penalty*3. This leaves six other problems on which at least one of the methods failed (see Table 6.6).

6.4. RHRL compared with automatic column scaling. Next we compare RHRL and Algorithm ACS proposed by Lalee and Nocedal [15]. ACS requires storage for an $n \times n$ lower-Hessenberg matrix plus $\mathcal{O}(n)$ additional locations; however, the implementation uses $n^2 + \mathcal{O}(n)$ elements, as does NPSOL. The flop count for ACS is not given by Lalee and Nocedal, but we estimate it to be $4n^2 + \mathcal{O}(n)$. This number is obtained as follows. A total of $\frac{3}{2}n^2 + \mathcal{O}(n)$ flops are required to restore the lower-

TABLE 6.6
Final nonoptimal gradients for RHRL and NPSOL on 6 CUTE problems.

Problem	RHRL (R3)	NPSOL
bdqrtic	1.3e-5	–
*engval*1	–	1.8e-6
fletchbv	–	1.1E+6
freuroth	3.6e-6	–
*penalty*2	–	2.6E+4
vardim	2.1e-5	–

TABLE 6.7
RHRL vs. ACS on 57 CUTE problems.

Method	Itn	Fcn	Cpu
RHRL (R3)	24667	34947	20:19
ACS (23)	32828	39725	29:38

Hessenberg matrix to a lower-triangular matrix L_k prior to solving for the search direction. Another n^2 flops are required to compute the search direction p_k. After some additional $\mathcal{O}(n)$ operations, $\frac{3}{2}n^2+\mathcal{O}(n)$ flops are required for the BFGS update, assuming that $L_k^T p_k$ is saved while computing p_k. Note that the work is essentially the same as that needed for NPSOL because both methods require two sweeps of rotations to maintain a triangular factor of the approximate Hessian. We have neglected any computations required for scaling since the version of ACS we tested scales very conservatively. In particular, the ACS code has six built-in rescaling strategies numbered 21–26. The last two only rescale during the first iteration. Option 23 appears to be the one preferred by Lalee and Nocedal since it performs the best on the problems of Moré, Garbow, and Hillstrom [18] (see Lalee and Nocedal [15, p. 20]). This is the option used in the tests below.

In our comparison, both RHRL and ACS meet the termination criteria on 57 of the 76 problems. In Table 6.7, we show the total numbers of iterations, function evaluations and cpu time for RHRL with R3 and for ACS with scaling option 23. Both methods fail on problems *arglinb*, *arglinc*, *bdqrtic*, *freuroth*, *hydc*20*ls*, *mancino*, *nonmsqrt*, and *penalty*3. This leaves nine other problems on which at least one of the methods fails (see Table 6.8).

6.5. RHRL compared with conjugate-direction scaling. In this section, we provide a comparison between RHRL and Algorithm CDS proposed by Siegel [28]. CDS requires $n^2+\mathcal{O}(n)$ storage locations, making it comparable with the implemented versions of both NPSOL and ACS. An iteration of the algorithm presented by Siegel [28, p. 9] requires $7n^2+n(n-l_c)+\mathcal{O}(n)$ flops when a "full" step is taken, where the parameter l_c is analogous to the partition parameter of RHRL. Otherwise the count is $4n^2+3nl_c+\mathcal{O}(n)$ flops (see [28, p. 23]). However, Siegel gives a more complicated formulation that requires only $3n^2+\mathcal{O}(n)$ flops per iteration (see [28, pp. 23–26]). For our comparison, the faster version of CDS was emulated by using the simpler formulation while counting the cpu time for only $3n^2+\mathcal{O}(n)$ flops per iteration. This was done as follows. In order to isolate the $3n^2$ flops, the flop count for the simpler CDS method was divided into five parts. The first part is the calculation of $V_k^T g_k$, which requires n^2 flops for both the full and partial step. The second part is the start of the Goldfarb–Powell BFGS update to V_k and the calculation of the search

TABLE 6.8
Final nonoptimal gradients for RHRL and ACS on 11 *CUTE problems.*

Problem	RHRL(R3)	ACS(23)
chainwoo	–	1.8e-6
cragglvy	–	2.7e-5
edensch	–	3.1e-6
*engval*1	–	2.7e-6
errinros	–	5.2e-6
*ncb*20	–	2.2e-6
*ncb*20*b*	–	4.5e-6
noncvxun	–	3.1e-6
*penalty*2	–	3.3e+1
tointgor	–	2.7e-6
vardim	2.1e-5	–

TABLE 6.9
*RHRL (R*2*) vs. CDS on* 68 *CUTE problems.*

Method	Itn	Fcn	Cpu
RHRL (R2)	27190	43577	22:20
CDS	26974	44003	27:10

direction. This part involves postmultiplying V_k by an orthogonal lower-Hessenberg matrix, Ω_k say, and requires $3n^2$ flops for the full step. (Powell [25, p. 42] suggests a way to reduce this cost.) In the case of the partial step, $3nl_c$ flops are required. In both cases, the search direction can be provided as a by-product at the same cost (see Powell [25, pp. 41–42]), but Siegel prefers to list this calculation separately. Hence, the third part of CDS is the calculation of the search direction, which requires n^2 and nl_c additional flops for the full and partial steps, respectively. The fourth part of CDS is the completion of the BFGS update, which requires an additional $2n^2$ flops for both steps (see Powell [25, p. 33]). The last part of CDS scales trailing columns of V_k and requires $n(n - l_c)$ flops (multiplications). Hence, in order to count only $3n^2$ flops per iteration for *both* types of step, we omit the cpu time for the three tasks of calculating $V_k\Omega_k$, computing the search direction, and scaling V_k.

The CDS code was implemented with the same line search used for RHR and RHRL. This allows a fair comparison of CDS with RHRL (R2), which is the reduced-Hessian variant satisfying the conditions of Theorem 4.3.

Table 6.9 illustrates the *connection* between RHRL and CDS (see Theorem 4.3) as well as the *advantage* of using the reduced-Hessian method. A direct comparison can be made because both methods meet the termination criteria on the same 68 problems. The problems on which both methods fail are *arglinb*, *arglinc*, *freuroth*, *hydc*20*ls*, *mancino*, *nonmsqrt*, *penalty*3, and *vardim*. Note that despite the similarity in the number of iterations and function evaluations, RHRL is roughly 21% faster than CDS. The improvement in cpu time is gained primarily because the reduced-Hessian approach allows the search direction to be computed more cheaply during iterations when r is much less than n.

To further illustrate the connection between RHRL (R2) and CDS, Table 6.10 compares data obtained for the two methods at particular iterations. This comparison is only for illustration and no statistical argument is being made. The three problems were chosen because the iterates match quite closely. Table 6.9 illustrates that the

TABLE 6.10
Iteration data for RHRL (R2) and CDS on 3 CUTE problems.

Problem	Method	k	α_k	$f(x_k)$	$\|g_k\|$	$\|g_k^T p_k\|$
broydn7d	RHRL	144	0.12E+00	0.12069659E+03	0.35E-05	0.19E-11
	CDS	144	0.12E+00	0.12069659E+03	0.35E-05	0.19E-11
dixmaanl	RHRL	322	0.10E+01	0.10000001E+01	0.57E-04	0.12E-06
	CDS	322	0.10E+01	0.10000001E+01	0.57E-04	0.12E-06
morebv	RHRL	300	0.10E+01	0.15708889E-07	0.71E-05	0.72E-08
	CDS	300	0.10E+01	0.15708889E-07	0.71E-05	0.72E-08

TABLE 6.11
RHRL (R3) vs. CDS on 67 CUTE problems.

Method	Itn	Fcn	Cpu
RHRL (R3)	26255	37082	21:10
CDS	26921	43900	27:07

iterates are not always identical.

When RHRL is used with R3, a further improvement in cpu time is gained relative to CDS. In this case, RHRL fails on one additional problem, *bdqrtic*, with final gradient norm 1.3×10^{-5}. Table 6.11 compares the iterations, function evaluations, and cpu time for the two methods on the set of 67/76 mutually successful test problems. Here, RHRL has a 28% advantage in cpu time.

7. Conclusions. Algorithms that compute an explicit reduced-Hessian approximation have two important advantages over conventional quasi-Newton methods. First, the amount of computation for each iteration is significantly less during the early stages. Second, approximate curvature along directions that lie off the manifold can be reinitialized as the iterations proceed, thereby reducing the influence of a poor initial estimate of the Hessian.

The results of section 6 indicate that reduced-Hessian methods can require substantially less computer time than a conventional BFGS method and some recently proposed extensions. Part of the reduction in computer time corresponds to the smaller number of iterations and function evaluations required when using the reinitialization strategy (see Tables 6.5, 6.7, 6.9, and 6.11). However, much of the reduction in computer time is the result of the average cost of an iteration being less than for competing methods. This result may seem surprising when it is considered that a reduced-Hessian iteration generally requires more work as the number of iterations approaches n. For example, if an RH direction is always used on a problem with dimension $n = 300$, an iteration of RHRL is more expensive than an iteration of CDS when $r_k \geq 170$. However, on 83% of the problems tested with dimension 300, the average value of r_k remains below this value. In most cases, the *maximum* value of r_k remained small relative to 170. Table 7.1 gives the average and maximum values of r_k for 54 CUTE problems with $n = 300$. The maximum value of r_k exceeds 170 on only 20 of the 54 problems listed, while the average value exceeds 170 on only 9 problems. (Remarkably, there are several cases where r_k does not exceed 50.) It is this feature that gives RHRL a significant advantage over the other algorithms tested in terms of the cost of the linear algebra per iteration.

TABLE 7.1
Final and average values of r_k on 54 CUTE problems with dimension $n = 300$.

Problem	Mean r	Final r	Problem	Mean r	Final r
arglina	2	2	*fletchbv*	288	300
arwhead	2	2	*fletchcr*	26	50
brownal	3	3	*genrose*	225	300
broydn7d	78	154	*hilberta*	9	12
brybnd	27	52	*hilbertb*	4	6
chainwoo	101	193	*liarwhd*	2	2
cosine	6	11	*morebv*	159	296
cragglvy	54	106	*ncb*20	87	173
dixmaana	3	3	*ncb*20b	166	300
dixmaanb	6	11	*noncvxu*2	164	298
dixmaanc	7	13	*noncvxun*	150	290
dixmaand	8	14	*nondia*	2	2
dixmaane	48	93	*nondquar*	217	300
dixmaanf	47	90	*penalty*1	2	2
dixmaang	46	91	*powellsg*	4	4
dixmaanh	45	88	*power*	86	98
dixmaani	170	300	*quartc*	274	298
dixmaank	173	300	*schmvett*	22	43
dixmaanl	169	300	*sinquad*	3	3
dixon3dq	158	300	*sparsine*	216	300
dqdrtic	5	5	*sparsqur*	174	300
dqrtic	274	298	*srosenbr*	2	2
edensch	14	29	*testquad*	104	161
*engval*1	12	23	*tointgss*	2	2
extrosnb	28	32	*tridia*	85	166
*fletcbv*2	149	297	*vareigvl*	104	204
*fletcbv*3	284	300	*woods*	4	4

Acknowledgments. We thank Marucha Lalee and Jorge Nocedal for graciously providing a copy of their ACS code.

REFERENCES

[1] I. BONGARTZ, A. R. CONN, N. I. M. GOULD, AND P. L. TOINT, CUTE*: Constrained and unconstrained testing environment*, ACM Trans. Math. Software, 21 (1995), pp. 123–160.

[2] J. W. DANIEL, W. B. GRAGG, L. KAUFMAN, AND G. W. STEWART, *Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization*, Math. Comput., 30 (1976), pp. 772–795.

[3] J. E. DENNIS, JR., AND R. B. SCHNABEL, *A new derivation of symmetric positive definite secant updates*, in Nonlinear Programming 4, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, London, New York, 1981, pp. 167–199.

[4] M. C. FENELON, *Preconditioned Conjugate-Gradient-Type Methods for Large-Scale Unconstrained Optimization*, Ph.D. thesis, Department of Operations Research, Stanford University, Stanford, CA, 1981.

[5] R. FLETCHER, *Practical Methods of Optimization*, 2nd ed., John Wiley, Chichester, New York, Brisbane, Toronto, Singapore, 1987.

[6] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization*, Computer Journal, 6 (1963), pp. 163–168.

[7] R. W. FREUND, G. H. GOLUB, AND N. M. NACHTIGAL, *Iterative solution of linear systems*, Acta Numer. 1992, Cambridge University Press, Cambridge, UK, 1992, pp. 57–100.

[8] P. E. GILL, G. H. GOLUB, W. MURRAY, AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comput., 28 (1974), pp. 505–535.

[9] P. E. GILL AND M. W. LEONARD, *Limited-Memory Reduced-Hessian Methods for Unconstrained Optimization*, Numerical Analysis Report NA 97-1, University of California, San Diego, CA, 1997.

[10] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *User's Guide for NPSOL (Version* 4.0*): A Fortran Package for Nonlinear Programming*, Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA, 1986.
[11] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, New York, 1981.
[12] D. Goldfarb, *Factorized variable metric methods for unconstrained optimization*, Math. Comput., 30 (1976), pp. 796–811.
[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.
[14] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, Frontiers in Appl. Math. 16, SIAM, Philadelphia, PA, 1995.
[15] M. Lalee and J. Nocedal, *Automatic column scaling strategies for quasi-Newton methods*, SIAM J. Optim., 3 (1993), pp. 637–653.
[16] M. W. Leonard, *Reduced Hessian Quasi-Newton Methods for Optimization*, Ph.D. thesis, Department of Mathematics, University of California, San Diego, CA, 1995.
[17] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Programming, 45 (1989), pp. 503–528.
[18] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.
[19] J. J. Moré and D. C. Sorensen, *Newton's method*, in Studies in Numerical Analysis, MAA Stud. Math. 24, G. H. Golub, ed., The Mathematical Association of America, Washington, DC, 1984, pp. 29–82.
[20] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Trans. Math. Software, 20 (1994), pp. 286–307.
[21] J. L. Nazareth, *The method of successive affine reduction for nonlinear minimization*, Math. Programming, 35 (1986), pp. 97–109.
[22] L. Nazareth, *A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms*, SIAM J. Numer. Anal., 16 (1979), pp. 794–800.
[23] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comput., 35 (1980), pp. 773–782.
[24] S. S. Oren and D. G. Luenberger, *Self-scaling variable metric (SSVM) algorithms, Part* I*: Criteria and sufficient conditions for scaling a class of algorithms*, Management Science, 20 (1974), pp. 845–862.
[25] M. J. D. Powell, *Updating conjugate directions by the BFGS formula*, Math. Programming, 38 (1987), pp. 693–726.
[26] D. F. Shanno and K. Phua, *Matrix conditioning and nonlinear optimization*, Math. Programming, 14 (1978), pp. 149–160.
[27] D. Siegel, *Implementing and Modifying Broyden Class Updates for Large Scale Optimization*, Report DAMTP/1992/NA12, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK, 1992.
[28] D. Siegel, *Modifying the BFGS update by a new column scaling technique*, Math. Programming, 66 (1994), pp. 45–78.