BFGS WITH UPDATE SKIPPING AND VARYING MEMORY

TAMARA GIBSON[†], DIANNE P. O'LEARY[‡], AND LARRY NAZARETH[§]

July 9, 1996

Abstract. We give conditions under which limited-memory quasi-Newton methods with exact line searches will terminate in n steps when minimizing n-dimensional quadratic functions. We show that although all Broyden family methods terminate in n steps in their full-memory versions, only BFGS does so with limited-memory. Additionally, we show that full-memory Broyden family methods with exact line searches terminate in at most $n + p$ steps when p matrix updates are skipped. We introduce new limited-memory BFGS variants and test them on nonquadratic minimization problems.

Key words. minimization, quasi-Newton, BFGS, limited-memory, update skipping, Broyden family

1. Introduction. The quasi-Newton family of algorithms remains a standard workhorse for minimization. Many of these methods share the properties of finite termination on strictly convex quadratic functions, a linear or superlinear rate of convergence on general convex functions, and no need to store or evaluate the second derivative matrix. In general, an approximation to the second derivative matrix is built by accumulating the results of earlier steps. Descriptions of many quasi-Newton algorithms can be found in books by Luenberger [16], Dennis and Schnabel [7], and Golub and Van Loan [11].

Although there are an infinite number of quasi-Newton methods, one method surpasses the others in popularity: the BFGS algorithm of Broyden, Fletcher, Goldfarb, and Shanno; see, e.g., Dennis and Schnabel [7]. This method exhibits more robust behavior than its relatives. Many attempts have been made to explain this robustness, but a complete understanding is yet to be obtained [23]. One result of the work in this paper is a small step toward this understanding, since we investigate the question of how much and which information can be dropped in BFGS and other quasi-Newton methods without destroying the property of quadratic termination.

We answer this question in the context of exact line search methods, those that find a minimizer on a one-dimensional subspace at every iteration. (In practice, inexact line searches that satisfy side conditions such as those proposed by Wolfe, see §4.3, are substituted for exact line searches.) We focus on modifications of well-known quasi-Newton algorithms resulting from limiting the memory, either by discarding the results of early steps (§2) or by skipping some updates to the second derivative approximation (§3). We give conditions under which quasi-Newton methods will terminate in n steps when minimizing quadratic functions of n variables. Although all Broyden family methods (see §2) terminate in n steps in their full-memory versions, we show that only BFGS has n-step termination under limited-memory. We also show that the methods from the Broyden family terminate in $n + p$ steps even if p updates are skipped, but termination is lost if we both skip updates and limit the memory.

[†]Applied Mathematics Program, University of Maryland, College Park, MD 20742. gibson@math.umd.edu. This work was supported in part by the National Physical Science Consortium, the National Security Agency, and the University of Maryland.

[‡]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. oleary@cs.umd.edu. This work was supported by the National Science Foundation under grant NSF 95-03126.

[§]Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164. nazareth@amath.washington.edu.

In §4, we report the results of experiments with new limited-memory BFGS variants on problems taken from the CUTE [3] test set, showing that some savings in time can be achieved.

Notation. Matrices and vectors are denoted by boldface upper-case and lower-case letters respectively. Scalars are denoted by Greek or Roman letters. The superscript "T" denotes transpose. Subscripts denote iteration number. Products are always taken from left to right. The notation $\text{span}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ denotes the subspace spanned by the vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$. Whenever we refer to an n-dimensional strictly convex quadratic function, we assume it is of the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b},$$

where $\mathbf{A}$ is a positive definite $n \times n$ matrix and $\mathbf{b}$ is an n-vector.

2. Limited-Memory Variations of Quasi-Newton Algorithms. In this section we characterize full-memory and limited-memory methods that terminate in n iterations on n-dimensional strictly convex quadratic minimization problems using exact line searches. Most full-memory versions of the methods we will discuss are known to terminate in n iterations. Limited-memory BFGS (L-BFGS) was shown by Nocedal [22] to terminate in n steps. The preconditioned conjugate gradient method, which can be cast as a limited-memory quasi-Newton method, is also known to terminate in n iterations; see, e.g., Luenberger [16]. Little else is known about termination of limited-memory methods.

Let $f(\mathbf{x})$ denote the strictly convex quadratic function to be minimized, and let $\mathbf{g}(\mathbf{x})$ denote the gradient of f. We define $\mathbf{g}_k \equiv \mathbf{g}(\mathbf{x}_k)$, where $\mathbf{x}_k$ is the kth iterate. Let

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,$$

denote the change in the current iterate and

$$\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k,$$

denote the change in gradient.

Let $\mathbf{x}_0$ be the starting point, and let $\mathbf{H}_0$ be the initial inverse Hessian approximation.
For $k = 0, 1, \ldots$

1. Compute $\mathbf{d}_k = -\mathbf{H}_k\mathbf{g}_k$.
2. Choose $\alpha_k > 0$ such that $f(\mathbf{x}_k + \alpha\mathbf{d}_k) \geq f(\mathbf{x}_k + \alpha_k\mathbf{d}_k)$ for all $\alpha > 0$.
3. Set $\mathbf{s}_k = \alpha_k\mathbf{d}_k$.
4. Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
5. Compute $\mathbf{g}_{k+1}$.
6. Set $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.
7. Choose $\mathbf{H}_{k+1}$.

FIG. 2.1. *General Quasi-Newton Method*

We present a general result that characterizes quasi-Newton methods, see Figure 2.1, that terminate in n iterations. We restrict ourselves to methods with an update of the form

$$\mathbf{H}_{k+1} = \gamma_k\mathbf{P}_k^T\mathbf{H}_0\mathbf{Q}_k + \sum_{i=1}^{m_k}\mathbf{w}_{ik}\mathbf{z}_{ik}^T. \tag{2.1}$$

Here,

1. $\mathbf{H}_0$ is an $n \times n$ symmetric positive definite matrix that remains constant for all k, and γ_k is a nonzero scalar that can be thought of as an iterative rescaling of $\mathbf{H}_0$.

2. $\mathbf{P}_k$ is an $n \times n$ matrix that is the product of projection matrices of the form

$$\mathbf{I} - \frac{\mathbf{u}\mathbf{v}^T}{\mathbf{u}^T\mathbf{v}}, \tag{2.2}$$

where $\mathbf{u} \in \operatorname{span}\{\mathbf{y}_0, \ldots, \mathbf{y}_k\}$ and $\mathbf{v} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k+1}\}$, and $\mathbf{Q}_k$ is an $n \times n$ matrix that is the product of projection matrices of the same form where $\mathbf{u}$ is any n-vector and $\mathbf{v} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}$,

3. m_k is a nonnegative integer, $\mathbf{w}_{ik}$ $(i = 1, 2, \ldots, m_k)$ is any n-vector, and $\mathbf{z}_{ik}$ $(i = 1, 2, \ldots, m_k)$ is any vector in $\operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}$.

We refer to this form as the *general form*. The general form fits many known quasi-Newton methods, including the Broyden family and the limited-memory BFGS method. We do not assume that these quasi-Newton methods satisfy the secant condition,

$$\mathbf{H}_{k+1}\mathbf{y}_k = \mathbf{s}_k,$$

nor that H_{k+1} is positive definite and symmetric. Symmetric positive definite updates are desirable since this guarantees that the quasi-Newton method produces descent directions. Note that if the update is not positive definite, we may produce a $\mathbf{d}_k$ such that $\mathbf{d}_k^T\mathbf{s}_k > 0$ in which case we choose α_k over all *negative* α rather than all positive α.

Example. The method of steepest descent [16] fits the general form (2.1). For each k we define

$$\gamma_k = 1, \quad m_k = 0, \text{ and } \mathbf{P}_k = \mathbf{Q}_k = \mathbf{H}_0 = \mathbf{I}. \tag{2.3}$$

Note that neither $\mathbf{w}$ nor $\mathbf{z}$ vectors are specified since $m_k = 0$.

Example. The $(k+1)$st update for the conjugate gradient method with preconditioner $\mathbf{H}_0$ fits the general form (2.1) with

$$\gamma_k = 1, \quad m_k = 0, \quad \mathbf{P}_k = \mathbf{I} - \frac{\mathbf{y}_k\mathbf{s}_k^T}{\mathbf{s}_k^T\mathbf{y}_k}, \text{ and } \mathbf{Q}_k = \mathbf{I}. \tag{2.4}$$

Example. The L-BFGS update, see Nocedal [22], with limited-memory constant m can be written as

$$\mathbf{H}_{k+1} = \mathbf{V}_{k-m_k+1,k}^T\mathbf{H}_0\mathbf{V}_{k-m_k+1,k} + \sum_{i=k-m_k+1}^{k} \mathbf{V}_{i+1,k}^T \frac{\mathbf{s}_i\mathbf{s}_i^T}{\mathbf{s}_i^T\mathbf{y}_i}\mathbf{V}_{i+1,k}, \tag{2.5}$$

where $m_k = \min\{k+1, m\}$ and

$$\mathbf{V}_{ik} = \prod_{j=i}^{k}\left(\mathbf{I} - \frac{\mathbf{y}_i\mathbf{s}_i^T}{\mathbf{s}_i^T\mathbf{y}_i}\right).$$

L-BFGS fits the general form (2.1) if at iteration k we choose

$$\begin{aligned} &\gamma_k = 1, \quad m_k = \min\{k+1, m\}, \\ &\mathbf{P}_k = \mathbf{Q}_k = \mathbf{V}_{k-m_k+1,k}, \text{ and} \\ &\mathbf{w}_{ik} = \mathbf{z}_{ik} = \frac{(\mathbf{V}_{k-m_k+i+1,k})^T(\mathbf{s}_{k-m_k+i})}{\sqrt{(\mathbf{s}_{k-m_k+i})^T(\mathbf{y}_{k-m_k+i})}}. \end{aligned} \tag{2.6}$$

Observe that $\mathbf{P}_k, \mathbf{Q}_k$ and $\mathbf{z}_{ik}$ all obey the constraints imposed on their construction.

BFGS is related to L-BFGS is the following way: if we were to use every $(\mathbf{s}, \mathbf{y})$ pair in the formation of each update (i.e. we have unlimited memory), we would be creating the same updates as BFGS. In practice, however, one would never do that because it would take more memory than storing the BFGS matrix.

Example. We will define limited-memory DFP (L-DFP). Our definition is consistent with the definition of limited-memory BFGS given in Nocedal [22]. Let $m \geq 1$ and let $m_k = \min\{k+1, m\}$. In order to define the L-DFP update we need to create a sequence of auxiliary matrices for $i = 0, \ldots, m_k$.

$$\begin{aligned}
\hat{\mathbf{H}}_{k+1}^{(0)} &= \mathbf{H}_0, \text{ and} \\
\hat{\mathbf{H}}_{k+1}^{(i)} &= \hat{\mathbf{H}}_{k+1}^{(i-1)} + \mathbf{U}_{\text{DFP}}(\hat{\mathbf{H}}_{k+1}^{(i-1)}, \mathbf{s}_{k-m_k+i}, \mathbf{y}_{k-m_k+i}),
\end{aligned}$$

where

$$\mathbf{U}_{\text{DFP}}(\mathbf{H}, \mathbf{s}, \mathbf{y}) = -\frac{\mathbf{H}\mathbf{y}\mathbf{y}^T\mathbf{H}}{\mathbf{y}^T\mathbf{H}\mathbf{y}} + \frac{\mathbf{s}\mathbf{s}^T}{\mathbf{s}^T\mathbf{y}}.$$

The matrix $\hat{\mathbf{H}}_{k+1}^{(m_k)}$ is the result of applying the DFP update m_k times to the matrix $\mathbf{H}_0$ with the m_k most recent $(\mathbf{s}, \mathbf{y})$ pairs. Thus, the $(k+1)$st L-DFP matrix is given by

$$\mathbf{H}_{k+1} = \hat{\mathbf{H}}_{k+1}^{(m_k)}.$$

To simplify our description, note that $\hat{\mathbf{H}}_{k+1}^{(i)}$ can be rewritten as

$$\begin{aligned}
\hat{\mathbf{H}}_{k+1}^{(i)} &= \left(\mathbf{I} - \frac{\hat{\mathbf{H}}_{k+1}^{(i-1)}\mathbf{y}_{k-m_k+i}\mathbf{y}_{k-m_k+i}^T}{\mathbf{y}_{k-m_k+i}^T\hat{\mathbf{H}}_{k+1}^{(i-1)}\mathbf{y}_{k-m_k+i}}\right)\hat{\mathbf{H}}_{k+1}^{(i-1)} + \frac{\mathbf{s}_{k-m_k+i}\mathbf{s}_{k-m_k+i}^T}{\mathbf{s}_{k-m_k+i}^T\mathbf{y}_{k-m_k+i}} \\
&= \left(\hat{\mathbf{V}}_{0k}^{(i)}\right)^T\mathbf{H}_0 + \sum_{j=1}^{i}\left(\hat{\mathbf{V}}_{jk}^{(i)}\right)^T\frac{\mathbf{s}_{k-m_k+j}\mathbf{s}_{k-m_k+j}^T}{\mathbf{s}_{k-m_k+j}^T\mathbf{y}_{k-m_k+j}},
\end{aligned}$$

for $i \geq 1$ where

$$\hat{\mathbf{V}}_{jk}^{(i)} = \prod_{l=j+1}^{i}\left[\mathbf{I} - \frac{\mathbf{y}_{k-m_k+l}\left(\mathbf{H}_{k+1}^{(l-1)}\mathbf{y}_{k-m_k+l}\right)^T}{\mathbf{y}_{k-m_k+l}^T\mathbf{H}_{k+1}^{(l-1)}\mathbf{y}_{k-m_k+l}}\right].$$

Thus $\mathbf{H}_{k+1}$ can be written as

$$\mathbf{H}_{k+1} = \mathbf{V}_{0k}^T\mathbf{H}_0 + \sum_{i=1}^{m_k}\left(\mathbf{V}_{ik}^T\frac{\mathbf{s}_{k-m_k+i}\mathbf{s}_{k-m_k+i}^T}{\mathbf{s}_{k-m_k+i}^T\mathbf{y}_{k-m_k+i}}\right), \tag{2.7}$$

where

$$\mathbf{V}_{ik} = \prod_{j=i+1}^{m_k}\left[\mathbf{I} - \frac{\mathbf{y}_{k-m_k+j}\left(\hat{\mathbf{H}}_{k+1}^{(j-1)}\mathbf{y}_{k-m_k+j}\right)^T}{\mathbf{y}_{k-m_k+j}^T\hat{\mathbf{H}}_{k+1}^{(j-1)}\mathbf{y}_{k-m_k+j}}\right].$$

Equation (2.7) looks very much like the general form given in (2.1). L-DFP fits the general form with the following choices:

$$\begin{aligned}(2.8)\qquad & \gamma_k = 1, \quad \mathbf{P}_k = \mathbf{V}_{0k}, \quad \mathbf{Q}_k = \mathbf{I}, \\ & \mathbf{w}_{ik} = \mathbf{V}_{ik}^T \mathbf{s}_{k-m_k+i} / (\mathbf{s}_{k-m_k+i}^T \mathbf{y}_{k-m_k+i}), \text{ and } \mathbf{z}_{ik} = \mathbf{s}_{k-m_k+i}.\end{aligned}$$

Except for the choice of $\mathbf{P}_k$, it is trivial to verify that the choices satisfy the general form. To prove that $\mathbf{P}_k$ satisfies the requirements, we need to show

$$(2.9)\qquad \hat{\mathbf{H}}_{k+1}^{(i-1)} \mathbf{y}_{k-m_k+i} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k+1}\}, \text{ for } i = 1, \ldots, m_k \text{ and all } k.$$

Proposition 2.1. *For limited-memory DFP, the following two conditions hold for each value of k:*

$$\begin{aligned}(2.10)\qquad & \hat{\mathbf{H}}_{k+1}^{(i-1)} \mathbf{y}_{k-m_k+i} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\} \textit{ for } i = 1, \ldots, m_k - 1 \textit{ and} \\ & \hat{\mathbf{H}}_{k+1}^{(i-1)} \mathbf{y}_{k-m_k+i} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k, \mathbf{H}_0 \mathbf{g}_{k+1}\} \textit{ for } i = m_k, \textit{ and} \\ (2.11)\qquad & \operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{k+1}\} \subseteq \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k+1}\}.\end{aligned}$$

Proof. We will prove this via induction. Suppose $k = 0$. Then $m_0 = 1$. We have

$$\hat{\mathbf{H}}_{k+1}^{(0)} \mathbf{y}_k = \mathbf{H}_0 \mathbf{y}_0 = \mathbf{H}_0 \mathbf{g}_1 - \mathbf{H}_0 \mathbf{g}_0 \in \operatorname{span}\{\mathbf{s}_0, \mathbf{H}_0 \mathbf{g}_1\}.$$

(Recall that $\operatorname{span}\{\mathbf{s}_0\}$ is trivially equal to $\operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0\}$.) Furthermore,

$$\begin{aligned}\mathbf{s}_1 &= -\alpha_1 \mathbf{H}_1 \mathbf{g}_1 \\ &= -\alpha_1 \left[\mathbf{H}_0 \mathbf{g}_1 - \frac{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{g}_1}{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{y}_0} (\mathbf{H}_0 \mathbf{g}_1 - \mathbf{H}_0 \mathbf{g}_0) + \frac{\mathbf{s}_0^T \mathbf{g}_1}{\mathbf{y}_0^T \mathbf{s}_0} \mathbf{s}_0 \right].\end{aligned}$$

So we can conclude,

$$\left(1 - \frac{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{g}_1}{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{y}_0}\right) \mathbf{H}_0 \mathbf{g}_1 = -\left[\frac{1}{\alpha_1} \mathbf{s}_1 + \frac{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{g}_1}{\mathbf{y}_0^T \mathbf{H}_0 \mathbf{y}_0} \mathbf{H}_0 \mathbf{g}_0 + \frac{\mathbf{s}_0^T \mathbf{g}_1}{\mathbf{y}_0^T \mathbf{s}_0} \mathbf{s}_0 \right].$$

Hence, $\mathbf{H}_0 \mathbf{g}_1 \in \operatorname{span}\{\mathbf{s}_0, \mathbf{s}_1\}$, and so the base case holds.

Assume that

$$\begin{aligned}& \hat{\mathbf{H}}_k^{(i-1)} \mathbf{y}_{k-1-m_{k-1}+i} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k-1}\} \text{ for } i = 1, \ldots, m_{k-1} - 1, \text{ and} \\ & \hat{\mathbf{H}}_k^{(i-1)} \mathbf{y}_{k-1-m_{k-1}+i} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k-1}, \mathbf{H}_0 \mathbf{g}_k\} \text{ for } i = m_{k-1}, \text{ and} \\ & \operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_k\} \subseteq \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}.\end{aligned}$$

We will use induction on i to show (2.10) for the $(k+1)$st case. For $i = 1$,

$$\hat{\mathbf{H}}_{k+1}^{(0)} \mathbf{y}_{k-m_k+1} = \mathbf{H}_0 \mathbf{y}_{k-m_k+1} = \mathbf{H}_0 \mathbf{g}_{k-m_k+2} - \mathbf{H}_0 \mathbf{g}_{k-m_k+1}.$$

Using the induction assumptions from the induction on k, we get that

$$\begin{aligned}& \hat{\mathbf{H}}_{k+1}^{(0)} \mathbf{y}_{k-m_k+1} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k, \mathbf{H}_0 \mathbf{g}_{k+1}\}, && \text{if } m_k = 1, \\ & \hat{\mathbf{H}}_{k+1}^{(0)} \mathbf{y}_{k-m_k+1} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}, && \text{otherwise.}\end{aligned}$$

Assume that $\hat{\mathbf{H}}_{k+1}^{(i-2)}\mathbf{y}_{k-m_k+i-1} \in \mathrm{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_k\}$ (induction assumption for i). Next,

$$\begin{aligned}\hat{\mathbf{H}}_{k+1}^{(i-1)}\mathbf{y}_{k-m_k+i} &= \left(\hat{\mathbf{V}}_{0k}^{(i-1)}\right)^T \mathbf{H}_0\mathbf{y}_{k-m_k+i} \\ &\quad + \sum_{j=1}^{i-1} \frac{\mathbf{s}_{k-m_k+j-1}^T\mathbf{y}_{k-m_k+i}}{\mathbf{s}_{k-m_k+j-1}^T\mathbf{y}_{k-m_k+j-1}} \left(\hat{\mathbf{V}}_{jk}^{(i-1)}\right)^T \mathbf{s}_{k-m_k+j-1}.\end{aligned}$$

For values of $i \le m_k - 1$, $\left(\hat{\mathbf{V}}_{jk}^{(i-1)}\right)^T$ maps any vector $\mathbf{v}$ into

$$\mathrm{span}\{\mathbf{v}, \hat{\mathbf{H}}_{k+1}^{(0)}\mathbf{y}_{k-m_k+1},\ldots,\hat{\mathbf{H}}_{k+1}^{(i-2)}\mathbf{y}_{k-2}\}.$$

and so $\hat{\mathbf{H}}_{k+1}^{(i-1)}\mathbf{y}_{k-m_k+i}$ is in

$$\mathrm{span}\{\mathbf{H}_0\mathbf{y}_{k-m_k+i}, \hat{\mathbf{H}}_{k+1}^{(0)}\mathbf{y}_{k-m_k+1},\ldots,\hat{\mathbf{H}}_{k+1}^{(i-2)}\mathbf{y}_{k-2}, \mathbf{s}_{k-m_k+1},\ldots,\mathbf{s}_{k-2}\}.$$

Using the induction assumptions for both i and k, we get

$$\hat{\mathbf{H}}_{k+1}^{(i-1)}\mathbf{y}_{k-m_k+i} \in \mathrm{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_k\},$$

and we can continue the induction on i. If $i = m_k$, then

$$\hat{\mathbf{H}}_{k+1}^{(m_k-1)}\mathbf{y}_k \in \mathrm{span}\{\mathbf{H}_0\mathbf{y}_k, \hat{\mathbf{H}}_{k+1}^{(0)}\mathbf{y}_{k-m_k+1},\ldots,\hat{\mathbf{H}}_{k+1}^{(m_k-2)}\mathbf{y}_{k-1}, \mathbf{s}_{k-m_k+1},\ldots,\mathbf{s}_{k-1}\},$$

so

$$\hat{\mathbf{H}}_{k+1}^{(m_k-1)}\mathbf{y}_k \in \mathrm{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_k, \mathbf{H}_0\mathbf{g}_{k+1}\}.$$

Hence the induction on i is complete and this proves (2.10) in the $(k+1)$st case.

Now, consider

$$\begin{aligned}\mathbf{s}_{k+1} &= -\alpha_{k+1}\mathbf{H}_{k+1}\mathbf{g}_{k+1} \\ &= \mathbf{V}_{0k}^T\mathbf{H}_0\mathbf{g}_{k+1} + \sum_{i=1}^{m_k} \frac{\mathbf{s}_{k-m_k+i}^T\mathbf{g}_{k+1}}{\mathbf{s}_{k-m_k+i}^T\mathbf{y}_{k-m_k+i}}\mathbf{V}_{ik}^T\mathbf{s}_{k-m_k+i}.\end{aligned}$$

Using the structure of $\mathbf{V}_{jk}$ and (2.10) we see that

$$\mathbf{H}_0\mathbf{g}_{k+1} \in \mathrm{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{k+1}\}.$$

Hence, (2.11) also holds in the $(k+1)$st case. $\square$

Example. The Broyden Class or Broyden Family is the class of quasi-Newton methods whose matrices are linear combinations of the DFP and BFGS matrices:

$$\mathbf{H}_{k+1} = \phi\,\mathbf{H}_k^{BFGS} + (1-\phi)\,\mathbf{H}_k^{DFP}, \qquad \phi \in \mathbf{R};$$

see, e.g., Luenberger [16, Chap. 9]. The parameter ϕ is usually restricted to values that are guaranteed to produce a positive definite update, although recent work with SR1, a Broyden Class method, by Khalfan, Byrd and Schnabel [14] may change this practice. No restriction on ϕ is necessary for the development of our theory. The Broyden class update can be expressed as

$$\begin{aligned}\mathbf{H}_{k+1} &= \mathbf{H}_k + \frac{\mathbf{s}_k\mathbf{s}_k}{\mathbf{s}_k\mathbf{y}_k} - \frac{\mathbf{H}_k\mathbf{y}_k\mathbf{y}_k^T\mathbf{H}_k}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k} \\ &\quad + \phi\,(\mathbf{y}_k\mathbf{H}_k\mathbf{y}_k)\left(\frac{\mathbf{s}_k}{\mathbf{s}_k^T\mathbf{y}_k} - \frac{\mathbf{H}_k\mathbf{y}_k}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k}\right)\left(\frac{\mathbf{s}_k}{\mathbf{s}_k^T\mathbf{y}_k} - \frac{\mathbf{H}_k\mathbf{y}_k}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k}\right)^T.\end{aligned}$$

We sketch the explanation of how the full-memory version fits the general form given in (2.1). The limited-memory case is similar. We can rewrite the Broyden Class update as follows:

$$\begin{aligned}
\mathbf{H}_{k+1} &= \mathbf{H}_k + (\phi - 1)\frac{\mathbf{H}_k\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k}\mathbf{H}_k - \phi\,\frac{\mathbf{s}_k\mathbf{y}_k^T}{\mathbf{s}_k^T\mathbf{y}_k}\mathbf{H}_k + \frac{\mathbf{s}_k\mathbf{s}_k^T}{\mathbf{s}_k^T\mathbf{y}_k} \\
&\quad + \phi\,\frac{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k\cdot\mathbf{s}_k\mathbf{s}_k^T}{(\mathbf{s}_k^T\mathbf{y}_k)^2} - \phi\,\frac{\mathbf{H}_k\mathbf{y}_k\mathbf{s}_k^T}{\mathbf{s}_k^T\mathbf{y}_k} \\
&= \left[\mathbf{I} - \frac{\left((1-\phi)\mathbf{s}_k^T\mathbf{y}_k\cdot\mathbf{H}_k\mathbf{y}_k + \phi\,\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k\cdot\mathbf{s}_k\right)\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k\cdot\mathbf{s}_k^T\mathbf{y}_k}\right]\mathbf{H}_k \\
&\quad + \left[\left(1 + \phi\,\frac{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k}{\mathbf{s}_k^T\mathbf{y}_k}\right)\mathbf{s}_k - \phi\,\mathbf{H}_k\mathbf{y}_k\right]\frac{\mathbf{s}_k^T}{\mathbf{s}_k^T\mathbf{y}_k}.
\end{aligned}$$

Hence,

$$\mathbf{H}_{k+1} = \mathbf{V}_{0k}\mathbf{H}_0 + \sum_{i=1}^{k+1}\mathbf{w}_{ik}\mathbf{z}_{ik}^T,$$

where

$$\begin{aligned}
\mathbf{V}_{ik} &= \textstyle\prod_{j=i}^{k}\left[\mathbf{I} - \frac{\left((1-\phi)\mathbf{s}_k^T\mathbf{y}_k\cdot\mathbf{H}_k\mathbf{y}_k + \phi\,\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k\cdot\mathbf{s}_k\right)\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{H}_k\mathbf{y}_k\cdot\mathbf{s}_k^T\mathbf{y}_k}\right], \\
\mathbf{w}_{ik} &= \mathbf{V}_{ik}\left[\left(1 + \phi\,\frac{\mathbf{y}_{i-1}^T\mathbf{H}_{i-1}\mathbf{y}_{i-1}}{\mathbf{s}_{i-1}^T\mathbf{y}_{i-1}}\mathbf{s}_{i-1}\right) - \phi\,\mathbf{H}_{i-1}\mathbf{y}_{i-1}\right], \text{ and } \mathbf{z}_{ik} = \frac{\mathbf{s}_{i-1}^T}{\mathbf{s}_{i-1}^T\mathbf{y}_{i-1}}.
\end{aligned}$$

It is left to the reader to show that $\mathbf{H}_k\mathbf{y}_k$ is in $\text{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{k+1}\}$, and thus the Broyden Class updates fit the form in (2.1).

2.1. Termination of Limited-Memory Methods. In this section we show that methods fitting the general form (2.1) produce conjugate search directions (Theorem 2.2) and terminate in n iterations (Corollary 2.3) *if and only if* $\mathbf{P}_k$ maps $\text{span}\{\mathbf{y}_0,\ldots,\mathbf{y}_k\}$ into $\text{span}\{\mathbf{y}_0,\ldots,\mathbf{y}_{k-1}\}$ for each $k = 1, 2, \ldots, n$. Furthermore, this condition on $\mathbf{P}_k$ is satisfied only if $\mathbf{y}_k$ is used in its formation (Corollary 2.4).

THEOREM 2.2. *Suppose that we apply a quasi-Newton method (Figure 2.1) with an update of the form (2.1) to minimize an n-dimensional strictly convex quadratic function. Then for each k before termination (i.e. $\mathbf{g}_{k+1} \neq 0$),*

$$\mathbf{g}_{k+1}^T\mathbf{s}_j = 0, \text{ for all } j = 0, 1, \ldots, k, \tag{2.12}$$

$$\mathbf{s}_{k+1}^T\mathbf{A}\mathbf{s}_j = 0, \text{ for all } j = 0, 1, \ldots, k, \text{ and} \tag{2.13}$$

$$\text{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{k+1}\} = \text{span}\{\mathbf{H}_0\mathbf{g}_0,\ldots,\mathbf{H}_0\mathbf{g}_{k+1}\}, \tag{2.14}$$

if and only if

$$\mathbf{P}_j\mathbf{y}_i \in \text{span}\{\mathbf{y}_0,\ldots,\mathbf{y}_{j-1}\}, \text{ for all } i = 0, 1, \ldots, j, \quad j = 0, 1, \ldots, k. \tag{2.15}$$

Proof. ($\Leftarrow$) Assume that (2.15) holds. We will prove (2.12)–(2.14) by induction. Since the line searches are exact, $\mathbf{g}_1$ is orthogonal to $\mathbf{s}_0$. Using the fact that $\mathbf{P}_0\mathbf{y}_0 = 0$

from (2.15), and the fact that $\mathbf{z}_{i0} \in \operatorname{span}\{\mathbf{s}_0\}$ implies $\mathbf{g}_1^T\mathbf{z}_{i0} = 0$, $i = 1,\ldots,m_k$, we see that $\mathbf{s}_1$ is conjugate to $\mathbf{s}_0$ since

$$\begin{aligned}
\mathbf{s}_1^T\mathbf{A}\mathbf{s}_0 &= \alpha_1\mathbf{d}_1^T\mathbf{y}_0 \\
&= -\alpha_1\mathbf{g}_1^T\mathbf{H}_1^T\mathbf{y}_0 \\
&= -\alpha_1\mathbf{g}_1^T\left(\gamma_0\mathbf{Q}_0^T\mathbf{H}_0\mathbf{P}_0 + \sum_{i=0}^{m_0}\mathbf{z}_{i0}\mathbf{w}_{i0}^T\right)\mathbf{y}_0 \\
&= -\alpha_1\left(\gamma_0\mathbf{g}_1^T\mathbf{Q}_0^T\mathbf{H}_0\mathbf{P}_0\mathbf{y}_0 + \sum_{i=0}^{m_0}\mathbf{g}_1^T\mathbf{z}_{i0}\mathbf{w}_{i0}^T\mathbf{y}_0\right) \\
&= 0.
\end{aligned}$$

Lastly, $\operatorname{span}\{\mathbf{s}_0\} = \operatorname{span}\{\mathbf{H}_0\mathbf{g}_0\}$, and so the base case is established.

We will assume that claims (2.12)–(2.14) hold for $k = 0,1,\ldots,\hat{k}-1$ and prove that they also hold for $k = \hat{k}$.

The vector $\mathbf{g}_{\hat{k}+1}$ is orthogonal to $\mathbf{s}_{\hat{k}}$ since the line search is exact. Using the induction hypotheses that $\mathbf{g}_{\hat{k}}$ is orthogonal to $\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}-1}\}$ and $\mathbf{s}_{\hat{k}}$ is conjugate to $\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}-1}\}$, we see that for $j < \hat{k}$,

$$\mathbf{g}_{\hat{k}+1}^T\mathbf{s}_j = (\mathbf{g}_{\hat{k}} + \mathbf{y}_{\hat{k}})^T\mathbf{s}_j = (\mathbf{g}_{\hat{k}} + \mathbf{A}\mathbf{s}_{\hat{k}})^T\mathbf{s}_j = 0.$$

Hence, (2.12) holds for $k = \hat{k}$.

To prove (2.13), we note that

$$\mathbf{s}_{\hat{k}+1}^T\mathbf{A}\mathbf{s}_j = -\alpha_{\hat{k}+1}\mathbf{g}_{\hat{k}+1}^T\mathbf{H}_{\hat{k}+1}^T\mathbf{y}_j,$$

so it is sufficient to prove that $\mathbf{g}_{\hat{k}+1}^T\mathbf{H}_{\hat{k}+1}^T\mathbf{y}_j = 0$ for $j = 0,1,\ldots,\hat{k}$. We will use the following facts:

(i) $\mathbf{g}_{\hat{k}+1}^T\mathbf{Q}_{\hat{k}}^T = \mathbf{g}_{\hat{k}+1}^T$ since the $\mathbf{v}$ in each of the projections used to form $\mathbf{Q}_{\hat{k}}$ is in $\operatorname{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}}\}$ and $\mathbf{g}_{\hat{k}+1}$ is orthogonal to that span.

(ii) $\mathbf{g}_{\hat{k}+1}^T\mathbf{z}_{i\hat{k}} = 0$ for $i = 1,\ldots,m_{\hat{k}}$ since each $\mathbf{z}_{i\hat{k}}$ is in $\operatorname{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}}\}$ and $\mathbf{g}_{\hat{k}+1}$ is orthogonal to that span.

(iii) Since we are assuming that (2.15) holds true, for each $j = 0,1,\ldots,\hat{k}$ there exists $\mu_0,\ldots,\mu_{\hat{k}-1}$ such that $\mathbf{P}_{\hat{k}}\mathbf{y}_j$ can be expressed as $\sum_{i=0}^{\hat{k}-1}\mu_i\mathbf{y}_i$.

(iv) For $i = 0,1,\ldots,\hat{k}-1$, $\mathbf{g}_{\hat{k}+1}$ is orthogonal to $\mathbf{H}_0\mathbf{y}_i$ because $\mathbf{g}_{\hat{k}+1}$ is orthogonal to $\operatorname{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}}\}$ and $\mathbf{H}_0\mathbf{y}_j \in \operatorname{span}\{\mathbf{s}_0,\ldots,\mathbf{s}_{\hat{k}}\}$ from (2.14).

Thus,

$$\begin{aligned}
\mathbf{g}_{\hat{k}+1}^T\mathbf{H}_{\hat{k}+1}^T\mathbf{y}_j &= \mathbf{g}_{\hat{k}+1}^T\left(\gamma_{\hat{k}}\mathbf{Q}_{\hat{k}}^T\mathbf{H}_0\mathbf{P}_{\hat{k}} + \sum_{i=1}^{m_{\hat{k}}}\mathbf{z}_{i\hat{k}}\mathbf{w}_{i\hat{k}}^T\right)\mathbf{y}_j \\
&= \gamma_{\hat{k}}\mathbf{g}_{\hat{k}+1}^T\mathbf{Q}_{\hat{k}}^T\mathbf{H}_0\mathbf{P}_{\hat{k}}\mathbf{y}_j + \sum_{i=1}^{m_{\hat{k}}}\mathbf{g}_{\hat{k}+1}^T\mathbf{z}_{i\hat{k}}\mathbf{w}_{i\hat{k}}^T\mathbf{y}_j \\
&= \gamma_{\hat{k}}\mathbf{g}_{\hat{k}+1}^T\mathbf{H}_0\mathbf{P}_{\hat{k}}\mathbf{y}_j \\
&= \gamma_{\hat{k}}\mathbf{g}_{\hat{k}+1}^T\mathbf{H}_0\left(\sum_{i=1}^{\hat{k}-1}\mu_i\mathbf{y}_i\right)
\end{aligned}$$

$$
\begin{aligned}
&= \gamma_{\hat{k}} \sum_{i=1}^{\hat{k}-1} \mu_i \mathbf{g}_{\hat{k}+1}^T \mathbf{H}_0 \mathbf{y}_i \\
&= 0.
\end{aligned}
$$

Thus, (2.13) holds for $k = \hat{k}$.

Lastly, using (i) and (ii) from above,

$$
\begin{aligned}
\mathbf{s}_{\hat{k}+1} &= -\alpha_{\hat{k}+1} \mathbf{H}_{\hat{k}+1} \mathbf{g}_{\hat{k}+1} \\
&= -\alpha_{\hat{k}+1} \left(\gamma_{\hat{k}} \mathbf{P}_{\hat{k}}^T \mathbf{H}_0 \mathbf{Q}_{\hat{k}} \mathbf{g}_{\hat{k}+1} + \sum_{i=1}^{m_{\hat{k}}} \mathbf{w}_{i\hat{k}} \mathbf{z}_{i\hat{k}}^T \mathbf{g}_{\hat{k}+1} \right) \\
&= -\alpha_{\hat{k}+1} \gamma_{\hat{k}} \mathbf{P}_{\hat{k}}^T \mathbf{H}_0 \mathbf{g}_{\hat{k}+1}.
\end{aligned}
$$

Since $\mathbf{P}_{\hat{k}}^T$ maps any vector $\mathbf{v}$ into $\operatorname{span}\{\mathbf{v}, \mathbf{s}_0, \ldots, \mathbf{s}_{\hat{k}+1}\}$ by construction, there exist $\sigma_0, \ldots, \sigma_{\hat{k}+1}$ such that

$$
\mathbf{s}_{\hat{k}+1} = -\alpha_{\hat{k}+1} \gamma_{\hat{k}} \left(\mathbf{H}_0 \mathbf{g}_{\hat{k}+1} + \sum_{i=0}^{\hat{k}+1} \sigma_i \mathbf{s}_i \right).
$$

Hence,

$$
\mathbf{H}_0 \mathbf{g}_{\hat{k}+1} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{\hat{k}+1}\},
$$

so

$$
\operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{\hat{k}+1}\} \subseteq \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{\hat{k}+1}\}.
$$

To show equality of the sets, we will show that $\mathbf{H}_0 \mathbf{g}_{\hat{k}+1}$ is linearly independent of $\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{\hat{k}}\}$. (We already know that the basis $\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{\hat{k}}\}$ is linearly independent since it spans the same space as the linearly independent set $\{\mathbf{s}_0, \ldots, \mathbf{s}_{\hat{k}}\}$ and has the same number of elements.) Suppose that $\mathbf{H}_0 \mathbf{g}_{\hat{k}+1}$ is not linearly independent. Then there exist $\phi_0, \ldots, \phi_{\hat{k}}$, not all zero, such that

$$
\mathbf{H}_0 \mathbf{g}_{\hat{k}+1} = \sum_{i=0}^{\hat{k}} \phi_i \mathbf{H}_0 \mathbf{g}_i.
$$

Recall that $\mathbf{g}_{\hat{k}+1}$ is orthogonal to $\{\mathbf{s}_0, \ldots, \mathbf{s}_{\hat{k}}\}$. By our induction assumption, this implies that $\mathbf{g}_{\hat{k}+1}$ is also orthogonal to $\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{\hat{k}}\}$. Thus for any j between 0 and $\hat{k}$,

$$
0 = \mathbf{g}_{\hat{k}+1}^T \mathbf{H}_0 \mathbf{g}_j = \left(\sum_{i=0}^{\hat{k}} \phi_i \mathbf{H}_0 \mathbf{g}_i \right)^T \mathbf{g}_j = \sum_{i=0}^{\hat{k}} \phi_i \mathbf{g}_i^T \mathbf{H}_0 \mathbf{g}_j = \phi_j \mathbf{g}_j^T \mathbf{H}_0 \mathbf{g}_j.
$$

Since $\mathbf{H}_0$ is positive definite and $\mathbf{g}_j$ is nonzero, we conclude that ϕ_j must be zero. Since this is true for every j between zero and k, we have a contradiction. Thus, the set $\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_{\hat{k}+1}\}$ is linearly independent. Hence, (2.14) holds for $k = \hat{k}$.

($\Rightarrow$) Assume that (2.12)–(2.14) hold for all k such that $\mathbf{g}_{k+1} \neq 0$ but that (2.15) does not hold; i.e., there exist j and k such that $g_{k+1} \neq 0$, j is between 0 and k, and

$$
\mathbf{P}_k \mathbf{y}_j \notin \operatorname{span}\{\mathbf{y}_0, \ldots, \mathbf{y}_{k-1}\} \tag{2.16}
$$

This will lead to a contradiction. By construction of $\mathbf{P}_k$, there exist $\mu_0, \ldots, \mu_k$ such that

$$\mathbf{P}_k \mathbf{y}_j = \sum_{i=0}^{k} \mu_i \mathbf{y}_i. \tag{2.17}$$

By assumption (2.16), μ_k must be nonzero. From (2.13), it follows that $\mathbf{g}_{k+1}^T \mathbf{H}_{k+1}^T \mathbf{y}_j = 0$. Using facts (i), (ii), and (iv) from before, (2.14) and (2.17), we get

$$\begin{aligned}
0 = \mathbf{g}_{k+1}^T \mathbf{H}_{k+1}^T \mathbf{y}_j &= \mathbf{g}_{k+1}^T \left(\gamma_k \mathbf{Q}_k^T \mathbf{H}_0 \mathbf{P}_k + \sum_{i=1}^{m_k} \mathbf{z}_{ik} \mathbf{w}_{ik}^T \right) \mathbf{y}_j \\
&= \gamma_k \mathbf{g}_{k+1}^T \mathbf{Q}_k^T \mathbf{H}_0 \mathbf{P}_k \mathbf{y}_j + \sum_{i=1}^{m_k} \mathbf{g}_{k+1}^T \mathbf{z}_{ik} \mathbf{w}_{ik}^T \mathbf{y}_j \\
&= \gamma_k \mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{P}_k \mathbf{y}_j \\
&= \gamma_k \mathbf{g}_{k+1}^T \mathbf{H}_0 \left(\sum_{i=0}^{k} \mu_i \mathbf{y}_i \right) \\
&= \gamma_k \sum_{i=0}^{k} \mu_i \mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{y}_i \\
&= \gamma_k \mu_k \mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{y}_k \\
&= \gamma_k \mu_k \left(\mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{g}_{k+1} - \mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{g}_k \right) \\
&= \gamma_k \mu_k \mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{g}_{k+1}.
\end{aligned}$$

Thus since neither γ_k nor μ_k is zero, we must have

$$\mathbf{g}_{k+1}^T \mathbf{H}_0 \mathbf{g}_{k+1} = 0,$$

but this is a contradiction since $\mathbf{H}_0$ is positive definite and $\mathbf{g}_{k+1}$ was assumed to be nonzero. □

When a method produces conjugate search directions, we can say something about termination.

COROLLARY 2.3. *Suppose we have a method of the type described in Theorem 2.2 satisfying (2.15). Suppose further that* $\mathbf{H}_j \mathbf{g}_j \neq 0$ *whenever* $g_j \neq 0$. *Then the scheme reproduces the iterates from the conjugate gradient method with preconditioner* $\mathbf{H}_0$ *and terminates in no more than* n *iterations.*

Proof. Let k be such that $\mathbf{g}_0, \ldots, \mathbf{g}_k$ are all nonzero and such that $\mathbf{H}_i \mathbf{g}_i \neq 0$ for $i = 0, \ldots, k$. Since we have a method of the type described in Theorem 2.2 satisfying (2.15), conditions (2.12) – (2.14) hold. We claim that the $(k+1)$st subspace of search directions, $\operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\}$, is equivalent to the $(k+1)$st Krylov subspace, $\operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, (\mathbf{H}_0 \mathbf{A})^k \mathbf{H}_0 \mathbf{g}_0\}$.

From (2.14), we know that $\operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_k\} = \operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_k\}$. We will show via induction that $\operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_k\} = \operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, (\mathbf{H}_0 \mathbf{A})^k \mathbf{H}_0 \mathbf{g}_0\}$. This base case is trivial, so assume that

$$\operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, \mathbf{H}_0 \mathbf{g}_i\} = \operatorname{span}\{\mathbf{H}_0 \mathbf{g}_0, \ldots, (\mathbf{H}_0 \mathbf{A})^i \mathbf{H}_0 \mathbf{g}_0\},$$

for some $i < k$. Now,

$$\mathbf{g}_{i+1} = \mathbf{A}\mathbf{x}_{i+1} - \mathbf{b} = \mathbf{A}(\mathbf{x}_i + \mathbf{s}_i) - \mathbf{b} = \mathbf{A}\mathbf{s}_i + \mathbf{g}_i,$$

and from (2.14) and the induction hypothesis,

$$\mathbf{s}_i \in \operatorname{span}\{\mathbf{H}_0\mathbf{g}_0, \ldots, \mathbf{H}_0\mathbf{g}_i\} = \operatorname{span}\{\mathbf{H}_0\mathbf{g}_0, \ldots, (\mathbf{H}_0\mathbf{A})^i\mathbf{H}_0\mathbf{g}_0\},$$

which implies that

$$\mathbf{H}_0\mathbf{A}\mathbf{s}_i \in \operatorname{span}\{(\mathbf{H}_0\mathbf{A})\mathbf{H}_0\mathbf{g}_0, \ldots, (\mathbf{H}_0\mathbf{A})^{i+1}\mathbf{H}_0\mathbf{g}_0\}.$$

So,

$$\mathbf{H}_0\mathbf{g}_{i+1} \in \operatorname{span}\{\mathbf{H}_0\mathbf{g}_0, \ldots, (\mathbf{H}_0\mathbf{A})^{i+1}\mathbf{H}_0\mathbf{g}_0\}.$$

Hence, the search directions span the Krylov subspace. Since the search directions are conjugate (2.13) and span the Krylov subspace, the iterates are the same as those produced by conjugate gradients with preconditioner $\mathbf{H}_0$.

Since we produce the same iterates as the conjugate gradient method and the conjugate gradient method is well-known to terminate within n iterations, we can conclude that this scheme terminates in at most n iterations. $\square$

Note that we require that $\mathbf{H}_j\mathbf{g}_j$ be nonzero whenever $\mathbf{g}_j$ is nonzero; this requirement is necessary since not all the methods produce positive definite updates and it is possible to construct an update that maps $\mathbf{g}_j$ to zero. If this were to happen, we would have a breakdown in the method.

The next corollary defines the role that the latest information ($\mathbf{s}_k$ and $\mathbf{y}_k$) plays in the formation of the kth $\mathbf{H}$-update.

COROLLARY 2.4. *Suppose we have a method of the type described in Theorem 2.2 satisfying (2.15). Suppose further that at the kth iteration* $\mathbf{P}_k$ *is composed of p projections of the form in (2.2). Then at least one of the projections must have* $\mathbf{u} = \sum_{i=0}^{k} \sigma_i \mathbf{y}_i$ *with* $\sigma_k \neq 0$. *Furthermore, if* $\mathbf{P}_k$ *is a single projection* ($p = 1$), *then* v *must be of the form* $\mathbf{v} = \rho_k \mathbf{s}_k + \rho_{k+1}\mathbf{s}_{k+1}$ *with* $\rho_k \neq 0$.

Proof. Consider the case of $p = 1$. We have

$$\mathbf{P}_k = \mathbf{I} - \frac{\mathbf{u}\mathbf{v}^T}{\mathbf{v}^T\mathbf{u}},$$

where $\mathbf{u} \in \operatorname{span}\{\mathbf{y}_0, \ldots, \mathbf{y}_k\}$ and $\mathbf{v} \in \operatorname{span}\{\mathbf{s}_0, \ldots, \mathbf{s}_{k+1}\}$. We will assume that

$$\mathbf{u} = \sum_{i=0}^{k} \sigma_i \mathbf{y}_i \quad \text{and } \mathbf{v} = \sum_{i=0}^{k+1} \rho_i \mathbf{s}_i.$$

for some scalars σ_i and ρ_i. By (2.15), there exist $\mu_0, \ldots, \mu_{k-1}$ such that

$$\mathbf{P}_k\mathbf{y}_k = \sum_{i=0}^{k-1} \mu_i \mathbf{y}_i.$$

Then

$$\mathbf{y}_k - \frac{\mathbf{v}^T\mathbf{y}_k}{\mathbf{v}^T\mathbf{u}}\mathbf{u} = \sum_{i=0}^{k-1} \mu_i \mathbf{y}_i,$$

and so

$$\frac{\mathbf{v}^T\mathbf{y}_k}{\mathbf{v}^T\mathbf{u}}\mathbf{u} = \mathbf{y}_k - \sum_{i=0}^{k-1} \mu_i \mathbf{y}_i. \tag{2.18}$$

From (2.13), the set $\{\mathbf{s}_0, \dots \mathbf{s}_k\}$ is conjugate and thus linearly independent. Since we are working with a quadratic, $\mathbf{y}_i = \mathbf{A}\mathbf{s}_i$ for all i; and since $\mathbf{A}$ is symmetric positive definite, the set $\{\mathbf{y}_0, \dots, \mathbf{y}_k\}$ is also linearly independent. So the coefficient of the $\mathbf{y}_k$ on the left-hand side of (2.18) must match that on the right-hand side, thus

$$\frac{\mathbf{v}^T\mathbf{y}_k}{\mathbf{v}^T\mathbf{u}}\sigma_k = 1.$$

Hence,

$$\sigma_k \neq 0, \tag{2.19}$$

and $\mathbf{y}_k$ must make a nontrivial contribution to $\mathbf{P}_k$.

Next we will show that $\rho_0 = \rho_1 = \cdots = \rho_{k-1} = 0$. Assume that j is between 0 and $k-1$. Then

$$\begin{aligned}
\mathbf{P}_k\mathbf{y}_j &= \mathbf{y}_j - \frac{\mathbf{v}^T\mathbf{y}_j}{\mathbf{v}^T\mathbf{u}}\mathbf{u} \\
&= \mathbf{y}_j - \frac{\left(\sum_{i=1}^{k+1}\rho_i\mathbf{s}_i\right)^T\mathbf{y}_j}{\mathbf{v}^T\mathbf{u}}\mathbf{u} \\
&= \mathbf{y}_j - \frac{\sum_{i=1}^{k+1}\rho_i\mathbf{s}_i^T\mathbf{A}\mathbf{s}_j}{\mathbf{v}^T\mathbf{u}}\mathbf{u} \\
&= \mathbf{y}_j - \frac{\rho_j\mathbf{s}_j^T\mathbf{A}\mathbf{s}_j}{\mathbf{v}^T\mathbf{u}}\mathbf{u}.
\end{aligned}$$

Now $\mathbf{s}_j\mathbf{A}\mathbf{s}_j$ is nonzero because $\mathbf{A}$ is positive definite. If ρ_j is nonzero then the coefficient of $\mathbf{u}$ is nonzero and so $\mathbf{y}_k$ must make a nontrivial contribution to $\mathbf{P}_k\mathbf{y}_j$, implying that $\mathbf{P}_k\mathbf{y}_j \notin \mathrm{span}\{\mathbf{y}_0, \dots, \mathbf{y}_{k-1}\}$. This is a contradiction. Hence, $\rho_j = 0$.

To show that $\rho_k \neq 0$, consider $\mathbf{P}_k\mathbf{y}_k$. Suppose that $\rho_k = 0$. Then

$$\begin{aligned}
\mathbf{v}^T\mathbf{y}_k &= \rho_{k+1}\mathbf{y}_k^T\mathbf{s}_{k+1} + \rho_k\mathbf{y}_k^T\mathbf{s}_k \\
&= \rho_{k+1}\mathbf{s}_k^T\mathbf{A}\mathbf{s}_{k+1} \\
&= 0,
\end{aligned}$$

and so

$$\mathbf{P}_k\mathbf{y}_k = \mathbf{y}_k - \frac{\mathbf{v}^T\mathbf{y}_k}{\mathbf{v}^T\mathbf{u}}\mathbf{u} = \mathbf{y}_k.$$

This contradicts $\mathbf{P}_k\mathbf{y}_k \in \mathrm{span}\{\mathbf{y}_0, \dots, \mathbf{y}_{k-1}\}$, so ρ_k must be nonzero.

Now we will discuss that $p > 1$ case. Label the $\mathbf{u}$-components of the p projections as $\mathbf{u}_1$ through $\mathbf{u}_p$. Then

$$\mathbf{P}_k\mathbf{y}_k = \mathbf{y}_k + \sum_{i=1}^{p}\gamma_i\mathbf{u}_i,$$

for some scalars γ_1 through γ_p. We know that

$$\mathbf{P}_k\mathbf{y}_k \in \mathrm{span}\{\mathbf{y}_0, \dots, \mathbf{y}_{k-1}\},$$

and that

$$\mathbf{y}_k \notin \text{span}\{\mathbf{y}_0, \ldots, \mathbf{y}_{k-1}\}.$$

Thus

$$\mathbf{y}_k \in \text{span}\{\mathbf{u}_1, \ldots, u_p\},$$

and since $\mathbf{u}_i \in \text{span}\{\mathbf{y}_0, \ldots, \mathbf{y}_k\}$ for $i = 1, \ldots, p$, we can conclude that at least one $\mathbf{u}_i$ must have a nontrivial contribution from $\mathbf{y}_k$. $\square$

2.2. Examples of Methods that Reproduce the CG Iterates. Here are some specific examples of methods that fit the general form, satisfy condition (2.15) of Theorem 2.2, and thus terminate in at most n iterations.

Example. The conjugate gradient method with preconditioner $\mathbf{H}_0$, see (2.4), satisfies condition (2.15) of Theorem 2.2 since

$$\mathbf{P}_k \mathbf{y}_j = \left(\mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} \right) \mathbf{y}_j = 0 \text{ for all } j = 0, \ldots, k.$$

Example. Limited-memory BFGS, see (2.6), satisfies condition (2.15) of Theorem 2.2 since

$$\mathbf{P}_k \mathbf{y}_j = \begin{cases} 0 & \text{for } j = k - m_k + 1, \ldots, k, \text{ and} \\ \mathbf{y}_j & \text{for } j = 0, \ldots, k - m_k. \end{cases}$$

Example. DFP (with full memory), see (2.8), satisfies condition (2.15) of Theorem 2.2. Consider $\mathbf{P}_k$ in the full memory case. We have

$$\mathbf{P}_k = \prod_{j=0}^{k} \left(\mathbf{I} - \frac{\mathbf{y}_i \mathbf{y}_i \mathbf{H}_i^T}{\mathbf{y}_i^T \mathbf{H}_i \mathbf{y}_i} \right).$$

For full-memory DFP, $\mathbf{H}_i \mathbf{y}_j = \mathbf{s}_j$ for $j = 0, \ldots, i-1$. Using this fact, one can easily verify that $\mathbf{P}_k \mathbf{y}_j = 0$ for $j = 0, \ldots, k$. Therefore, full-memory DFP satisfies condition (2.15) of Theorem 2.2. The same reasoning does not apply to the limited-memory case as we shall show in §2.3.

The next corollary gives some ideas for other methods that are related to L-BFGS and terminate in at most n iterations on strictly convex quadratics.

COROLLARY 2.5. *The L-BFGS (2.5) method will terminate in n iterations on an n-dimensional strictly convex quadratic function even if any combination of the following modifications is made to the update:*

1. *Vary the limited-memory constant, keeping $m_k \geq 1$.*
2. *Form the projections used in $\mathbf{V}_k$ from the most recent $(\mathbf{s}_k, \mathbf{y}_k)$ pair along with any set of $m-1$ other pairs from $\{(\mathbf{s}_0, \mathbf{y}_0), \ldots, (\mathbf{s}_{k-1}, \mathbf{y}_{k-1})\}$.*
3. *Form the projections used in $\mathbf{V}_k$ from the most recent $(\mathbf{s}_k, \mathbf{y}_k)$ pair along with any $m-1$ other linear combinations of pairs from $\{(\mathbf{s}_0, \mathbf{y}_0), \ldots, (\mathbf{s}_{k-1}, \mathbf{y}_{k-1})\}$.*
4. *Iteratively rescale $\mathbf{H}_0$.*

Proof. For each variant, we show that the method fits the general form in (2.1), satisfies condition (2.15) of Theorem 2.2 and hence terminates by Corollary 2.3.

1. Let $m > 0$ be any value which may change from iteration to iteration, and define

$$\mathbf{V}_{ik} = \prod_{j=i}^{k} \left(\mathbf{I} - \frac{\mathbf{y}_j \mathbf{s}_j^T}{\mathbf{s}_j^T \mathbf{y}_j} \right).$$

Choose

$$\begin{aligned}
&\gamma_k = 1, \quad m_k = \min\{k+1, m\}, \\
&\mathbf{P}_k = \mathbf{Q}_k = \mathbf{V}_{k-m_k+1,k}, \text{ and} \\
&\mathbf{w}_{ik} = \mathbf{z}_{ik} = \frac{(\mathbf{V}_{k-m_k+i+1,k})^T(\mathbf{s}_{k-m_k+i})}{\sqrt{(\mathbf{s}_{k-m_k+i})^T(\mathbf{y}_{k-m_k+i})}}.
\end{aligned}$$

These choices fit the general form. Furthermore,

$$\mathbf{P}_k\mathbf{y}_j = \begin{cases} 0 & \text{if } j = k-m_k, k-m_k+1, \ldots, k, \text{ and} \\ \mathbf{y}_j & \text{if } j = 0, 1, \ldots, k-m_k-1, \end{cases}$$

so this variation satisfies condition (2.15) of Theorem 2.2.

2. This is a special case of the next variant.

3. At iteration k, let $(\hat{\mathbf{s}}_k^{(i)}, \hat{\mathbf{y}}_k^{(i)})$ denote the ith ($i = 1, \ldots, m-1$) choice of any linear combination from the span of the set

$$\{(\mathbf{s}_0, \mathbf{y}_0), \ldots, (\mathbf{s}_{k-1}, \mathbf{y}_{k-1})\},$$

and let $(\hat{\mathbf{s}}_k^{(m)}, \hat{\mathbf{y}}_k^{(m)}) = (\mathbf{s}_k, \mathbf{y}_k)$. Define

$$\mathbf{V}_{ik} = \prod_{j=i}^{m}\left(\mathbf{I} - \frac{(\hat{\mathbf{y}}_k^{(i)})(\hat{\mathbf{s}}_k^{(i)})^T}{(\hat{\mathbf{s}}_k^{(i)})^T(\hat{\mathbf{y}}_k^{(i)})}\right).$$

Choose

$$\begin{aligned}
&\gamma_k = 1, \quad m_k = \min\{k+1, m\}, \\
&\mathbf{P}_k = \mathbf{Q}_k = \mathbf{V}_{1,k}, \text{ and} \\
&\mathbf{w}_{ik} = \mathbf{z}_{ik} = \frac{(\mathbf{V}_{i+1,k})^T(\hat{\mathbf{s}}_k^{(i)})}{\sqrt{(\hat{\mathbf{s}}_k^{(i)})^T(\hat{\mathbf{y}}_k^{(i)})}}.
\end{aligned}$$

These choices satisfy the general form (2.1). Furthermore,

$$\mathbf{P}_k\mathbf{y}_j = \begin{cases} 0 & \text{if } \mathbf{y}_j = \mathbf{y}_k^{(i)} \text{ for some } i, \text{ and} \\ \mathbf{y}_j & \text{otherwise.} \end{cases}$$

Hence, this variation satisfies condition (2.15) of Theorem 2.2.

4. Let γ_k in (2.1) be the scaling constant, and choose the other vectors and matrices as in L-BFGS (2.6).

Combinations of variants are left to the reader. □

Remark. Part 3 of the previous corollary shows that the "accumulated step" method of Gill and Murray [10] terminates on quadratics.

Remark. Part 4 of the previous corollary shows that scaling does not affect termination in L-BFGS. In fact, for any method that fits the general form, it is easy to see that scaling will not affect termination on quadratics.

2.3. Examples of Methods that Do Not Reproduce the CG Iterates. We will discuss several methods that fit the general form given in (2.1) but do not satisfy the conditions of Theorem 2.2.

Example. Steepest descent, see (2.3), does not satisfy condition (2.15) of Theorem 2.2 and thus does not produce conjugate search directions. This fact is well-known; see, e.g., Luenberger [16].

Example. Limited-memory DFP, see (2.8), with $m < n$ does not satisfy the condition on $\mathbf{P}_k$ (2.15) for all k, and so the method will not produce conjugate directions.

For example, suppose that we have a convex quadratic with

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Using a limited-memory constant of $m = 1$ and exact arithmetic, it can be seen that the iteration does not terminate within the first 20 iterations of limited-memory DFP with $\mathbf{H}_0 = \mathbf{I}$. The MAPLE notebook file used to compute this example is available on the World Wide Web [9].

Remark. Using the above example, we can easily see that no limited-memory Broyden class method except limited-memory BFGS terminates within the first n iterations.

3. Update-Skipping Variations for Broyden Class Quasi-Newton Algorithms. The previous section discussed limited-memory methods that behave like conjugate gradients on n-dimensional strictly convex quadratic functions. In this section, we are concerned with methods that skip some updates in order to reduce the memory demands. We establish conditions under which finite termination is preserved but delayed for the Broyden Class.

3.1. Termination when Updates are Skipped. It was shown by Powell [26] that if we skip every other update and take *direct prediction steps* (i.e. steps of length one) in a Broyden class method, then the procedure will terminate in no more than $2n+1$ iterations on an n-dimensional strictly convex quadratic function. An alternate proof of this result is given by Nazareth [21].

We will prove a related result. Suppose that we are doing *exact line searches* using a Broyden Class quasi-Newton method on a strictly convex quadratic function and decide to "skip" p updates to $\mathbf{H}$ (i.e. choose $\mathbf{H}_{k+1} = \mathbf{H}_k$ on p occasions). Then, the algorithm terminates in no more than $n + p$ iterations. In contrast to Powell's result, it does not matter which updates are skipped or if multiple updates are skipped in a row.

THEOREM 3.1. *Suppose that a Broyden Class method using exact line searches is applied to an n-dimensional strictly convex quadratic function and p updates are skipped. Let*

$$J(k) = \{j \le k : \textit{ the update at iteration } j \textit{ is not skipped}\}.$$

Then for all $k = 0, 1, \dots$

$$\mathbf{g}_{k+1}^T \mathbf{s}_j = 0, \textit{ for all } j \in J(k), \textit{ and} \tag{3.1}$$

$$\mathbf{s}_{k+1}^T \mathbf{A} \mathbf{s}_j = 0, \textit{ for all } j \in J(k). \tag{3.2}$$

Furthermore, the method terminates in at most $n + p$ iterations at the exact minimizer.

Proof. We will use induction on k to show (3.1) and

$$\mathbf{H}_{k+1} \mathbf{y}_j = \mathbf{s}_j, \text{ for all } j \in J(k). \tag{3.3}$$

Then (3.2) follows easily since for all $j \in J(k)$,

$$\begin{aligned} \mathbf{s}_{k+1}^T \mathbf{A} \mathbf{s}_j &= -\alpha_{k+1} \mathbf{g}_{k+1} \mathbf{H}_{k+1} \mathbf{y}_j \\ &= -\alpha_{k+1} \mathbf{g}_{k+1} \mathbf{s}_j \\ &= 0. \end{aligned}$$

Let k_0 be the least value of k such that $J(k)$ is nonempty; i.e., $J(k_0) = \{k_0\}$. Then $\mathbf{g}_{k_0+1}$ is orthogonal to $\mathbf{s}_{k_0}$ since line searches are exact, and $\mathbf{H}_{k_0+1}\mathbf{y}_{k_0} = \mathbf{s}_{k_0}$ since all members of the Broyden Family satisfy the secant condition. Hence, the base case is true. Now assume that (3.1) and (3.3) hold for all values of $k = 0, 1, \ldots, \hat{k}-1$. We will show that they also hold for $k = \hat{k}$.

Case I. Suppose that $\hat{k} \notin J(\hat{k})$. Then $\mathbf{H}_{\hat{k}+1} = \mathbf{H}_{\hat{k}}$ and $J(\hat{k}-1) = J(\hat{k})$, so for any $j \in J(\hat{k})$,

$$\begin{aligned} \mathbf{g}_{\hat{k}+1}^T \mathbf{s}_j &= (\mathbf{g}_{\hat{k}} + \mathbf{A}\mathbf{s}_{\hat{k}})^T \mathbf{s}_j \\ &= \mathbf{g}_{\hat{k}}^T \mathbf{s}_j + \mathbf{s}_{\hat{k}}^T \mathbf{A} \mathbf{s}_j \\ &= 0, \end{aligned} \tag{3.4}$$

and

$$\mathbf{H}_{\hat{k}+1}\mathbf{y}_j = \mathbf{H}_{\hat{k}}\mathbf{y}_j = \mathbf{s}_j.$$

Case II. Suppose that $\hat{k} \in J(\hat{k})$. Then $\mathbf{H}_{\hat{k}+1}$ satisfies the secant condition and $J(\hat{k}) = J(\hat{k}-1) \cup \{\hat{k}\}$. Now $\mathbf{g}_{\hat{k}+1}$ is orthogonal to $\mathbf{s}_k$ since the line searches are exact, and it is orthogonal to the older $\mathbf{s}_j$ by the argument in (3.4). The secant condition guarantees that $\mathbf{H}_{\hat{k}+1}\mathbf{y}_{\hat{k}} = \mathbf{s}_{\hat{k}}$, and for $j \in J(\hat{k})$ but $j \neq \hat{k}$ we have

$$\begin{aligned} \mathbf{H}_{\hat{k}+1}\mathbf{y}_j &= \mathbf{H}_{\hat{k}}\mathbf{y}_j + \frac{\mathbf{s}_{\hat{k}}\mathbf{s}_{\hat{k}}^T}{\mathbf{s}_{\hat{k}}\mathbf{y}_{\hat{k}}}\mathbf{y}_j - \frac{\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}\mathbf{y}_j \\ &\quad + \phi\left(\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}\right)\left(\frac{\mathbf{s}_{\hat{k}}}{\mathbf{s}_{\hat{k}}^T\mathbf{y}_{\hat{k}}} - \frac{\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}\right)\left(\frac{\mathbf{s}_{\hat{k}}}{\mathbf{s}_{\hat{k}}^T\mathbf{y}_{\hat{k}}} - \frac{\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}\right)^T\mathbf{y}_j \\ &= \mathbf{s}_j + \frac{\mathbf{s}_{\hat{k}}^T\mathbf{A}\mathbf{s}_j}{\mathbf{s}_{\hat{k}}\mathbf{y}_{\hat{k}}}\mathbf{s}_{\hat{k}} - \frac{\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}\mathbf{y}_{\hat{k}}^T\mathbf{s}_j}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}} \\ &\quad + \phi\left(\mathbf{y}_{\hat{k}}\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}\right)\left(\frac{\mathbf{s}_{\hat{k}}}{\mathbf{s}_{\hat{k}}^T\mathbf{y}_{\hat{k}}} - \frac{\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}\right)\left(\frac{\mathbf{s}_{\hat{k}}^T\mathbf{A}\mathbf{s}_j}{\mathbf{s}_{\hat{k}}^T\mathbf{y}_{\hat{k}}} - \frac{\mathbf{y}_{\hat{k}}^T\mathbf{s}_j}{\mathbf{y}_{\hat{k}}^T\mathbf{H}_{\hat{k}}\mathbf{y}_{\hat{k}}}\right) \\ &= \mathbf{s}_j. \end{aligned}$$

In either case, the induction result follows.

Suppose that we skip p updates. Then the set $J(n-1+p)$ has cardinality n. Without loss of generality, assume that the set $\{\mathbf{s}_i\}_{i\in J(n-1+p)}$ has no zero elements. From (3.2), the vectors are linearly independent. By (3.1),

$$\mathbf{g}_{n+p}^T\mathbf{s}_j = 0, \text{ for all } j \in J(n-1+p),$$

and so $\mathbf{g}_{n+p}$ must be zero. This implies that $\mathbf{x}_{n+p}$ is the exact minimizer of f. $\square$

3.2. Loss of Termination for Update Skipping with Limited-Memory. Unfortunately, updates that use both limited-memory and repeated update-skipping do not produce n conjugate search directions for n-dimensional strictly convex quadratics, and the termination property is lost. We will show a simple example, limited-memory BFGS with $m = 1$, skipping every other update. Note that according to Corollary 2.4, we would still be guaranteed termination if we used the most recent information in each update.

Example. Suppose that we have a convex quadratic with

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

We apply limited-memory BFGS with limited-memory constant $m = 1$ and $\mathbf{H}_0 = \mathbf{I}$ and skip every-other update to $\mathbf{H}$. Using exact arithmetic in MAPLE, we observe that the process does not terminate even after 100 iterations [9].

4. Experimental Results. The results of §2 and §3 lead to a number of ideas for new methods for unconstrained optimization. In this section, we motivate, develop, and test these ideas. We describe the collection of test problems in §4.2. The test environment is described in §4.3. Section 4.4.1 outlines the implementation of the L-BFGS method (our base for all comparisons) and §§4.4.2–4.4.7 describe the variations. Pseudo-code for L-BFGS and its variations is given in Appendix B. Complete numerical results, many graphs of the numerical results, and the original FORTRAN code are available [9].

4.1. Motivation. So far we have only given results for convex quadratic functions. While termination on quadratics is beautiful in theory, it does not necessarily yield insight into how these methods will do in practice.

We will not present any new results relating to convergence of these algorithms on general functions; however, many of these can be shown to converge using the convergence analysis presented in §7 of [15]. In [15], Liu and Nocedal show that a limited-memory BFGS method implemented with a line search that satisfies the strong Wolfe conditions (see §4.3 for a definition) is R-linearly convergent on a convex function that satisfies a few modest conditions.

4.2. Test Problems. For our test problems, we used the Constrained and Unconstrained Testing Environment (CUTE) by Bongartz, Conn, Gould and Toint. The package is documented in [3] and can be obtained via the world wide web [2] or via ftp [1]. The package contains a large collection of test problems as well as the interfaces necessary for using the problems. The test problems are stored as “SIF” files. We chose a collection of 22 unconstrained problems. The problems ranged in size from 10 to 10,000 variables, but each took L-BFGS with limited-memory constant $m = 5$ at least 60 iterations to solve. Table 4.1 enumerates the problems, giving the SIF file name, the dimension (n), and a description for each problem. The CUTE package also provides a starting point (x_0) for each problem.

4.3. Test Environment. We used FORTRAN77 code on an SGI Indigo^2 to run the algorithms, with FORTRAN BLAS routines from NETLIB. We used the compiler's default optimization level.

Figure 2.1 outlines the general quasi-Newton implementation that we followed. For the line search, we use the routines `cvsrch` and `cstep` written by Jorge J. Moré

No.	SIF Name	n	Description & Reference
1	EXTROSNB	10	Extended Rosenbrock function (nonseparable version) [30, Problem 10].
2	WATSONS	31	Watson problem [17, Problem 20].
3	TOINTGOR	50	Toint's operations research problem [29].
4	TOINTPSP	50	Toint's PSP operations research problem [29].
5	CHNROSNB	50	Chained Rosenbrock function [29].
6	ERRINROS	50	Nonlinear problem similar to CHNROSNB [28].
7	FLETCHBV	100	Fletcher's boundary value problem [8, Problem 1].
8	FLETCHCR	100	Fletcher's chained Rosenbrock function [8, Problem 2].
9	PENALTY2	100	Second penalty problem [17, Problem 24].
10	GENROSE	500	Generalized Rosenbrock function [18, Problem 5].
11	BDQRTIC	1000	Quartic with a banded Hessian with bandwidth=9 [5, Problem 61].
12	BROYDN7D	1000	Seven diagonal variant of the Broyden tridiagonal system with a band away from diagonal [29].
13	PENALTY1	1000	First penalty problem [17, Problem 23].
14	POWER	1000	Power problem by Oren [25].
15	MSQRTALS	1024	The dense matrix square root problem by Nocedal and Liu (case 0) seen as a nonlinear equation problem [4, Problem 204].
16	MSQRTBLS	1025	The dense matrix square root problem by Nocedal and Liu (case 1) seen as a nonlinear equation problem [4, Problem 201].
17	CRAGGLVY	5000	Extended Cragg & Levy problem [30, Problem 32].
18	NONDQUAR	10000	Nondiagonal quartic test problem [5, Problem 57].
19	POWELLSG	10000	Extended Powell singular function [17, Problem 13].
20	SINQUAD	10000	Another function with nontrivial groups and repetitious elements [12].
21	SPMSRTLS	10000	Liu and Nocedal tridiagonal matrix square root problem [4, Problem 151].
22	TRIDIA	10000	Shanno's TRIDIA quadratic tridiagonal problem [30, Problem 8].

TABLE 4.1
Test problem collection. Each problems was chosen from the CUTE package.

and David Thuente from a 1983 version of MINPACK. This line search routine finds an α that meets the strong Wolfe conditions,

$$f(\mathbf{x}+\alpha\mathbf{d}) \leq f(\mathbf{x})+\omega_1\alpha\mathbf{g}(\mathbf{x})^T\mathbf{d}, \tag{4.1}$$

$$|\mathbf{g}(\mathbf{x}+\alpha\mathbf{d})^T\mathbf{d}| \leq \omega_2|\mathbf{g}(\mathbf{x})^T\mathbf{s}|; \tag{4.2}$$

see, e.g., Nocedal [23]. We used $\omega_1 = 1.0\times10^{-4}$ and $\omega_2 = 0.9$. Except for the first iteration, we always attempt a step length of 1.0 first and only use an alternate value if 1.0 does not satisfy the Wolfe conditions. In the first iteration, we initially try a step length equal to $||\mathbf{g}_0||^{-1}$. The remaining line search parameters are detailed in Appendix A.

We generate the matrix $\mathbf{H}_k$ by either the limited-memory update or one of the variations described in §4.4, storing the matrix implicitly in order to save both memory and computation time.

We terminate the iterations if any of the following conditions are met at iteration

k:

1. The inequality

$$\frac{||\mathbf{g}_k||}{||\mathbf{x}_k||} < 1.0 \times 10^{-5},$$

is satisfied,

2. the line search fails, or
3. the number of iterations exceeds 3000.

We say that the iterates have converged if the first condition is satisfied. Otherwise, the method has failed.

4.4. L-BFGS and its variations. We tried a number of variations to the standard L-BFGS algorithm. L-BFGS and these variations are described in this subsection and summarized in Table 4.2.

4.4.1. L-BFGS: Algorithm 0. The limited-memory BFGS update is given in (2.5) and described fully by Byrd, Nocedal and Schnabel [22]. Our implementation and the following description come essentially from [22].

Let $\mathbf{H}_0$ be symmetric and positive definite and assume that the m_k pairs

$$\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-m_k}^{k-1}$$

each satisfy $\mathbf{s}_i^T\mathbf{y}_i > 0$.

We will let

$$\mathbf{S}_k = [\mathbf{s}_{k-m_k}\mathbf{s}_{k-m_k+1}\cdots\mathbf{s}_{k-1}] \text{ and } \mathbf{Y}_k = [\mathbf{y}_{k-m_k}\mathbf{y}_{k-m_k+1}\cdots\mathbf{y}_{k-1}],$$

where $m_k = \min\{k+1, m\}$, and m is some positive integer. We will assume that $\mathbf{H}_0 = \mathbf{I}$ and that $\mathbf{H}_0$ is iteratively rescaled by a constant γ_k as is commonly done in practice. Then, the matrix $\mathbf{H}_k$ obtained by k applications of the limited-memory BFGS update can be expressed as

$$\mathbf{H}_k = \gamma_k\mathbf{I} + \begin{pmatrix} \mathbf{S}_k & \gamma_k\mathbf{Y}_k \end{pmatrix} \begin{pmatrix} \mathbf{U}_k^{-T}(\mathbf{D}_k + \gamma_k\mathbf{Y}_k^T\mathbf{Y}_k)\mathbf{U}_k^{-1} & -\mathbf{U}_k^{-T} \\ -\mathbf{U}_k^{-1} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{S}_k^T \\ \gamma_k\mathbf{Y}_k^T \end{pmatrix},$$

where $\mathbf{U}_k$ and $\mathbf{D}_k$ are the $m_k \times m_k$ matrices given by

$$(\mathbf{U}_k)_{ij} = \begin{cases} (\mathbf{s}_{k-m_k-1+i})^T(\mathbf{y}_{k-m_k-1+j}) & \text{if } i \le j, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\mathbf{D}_k = \text{diag}\{(\mathbf{s}_{k-m_k})^T(\mathbf{y}_{k-m_k}), \ldots, \mathbf{s}_{k-1}^T\mathbf{y}_{k-1}\}.$$

We will describe how to compute $\mathbf{d}_k = -\mathbf{H}_k\mathbf{g}_k$ in the case that $k > 0$. Let $\mathbf{x}_k$ be the current iterate. Let $m_k = \min\{k+1, m\}$. Given $\mathbf{s}_{k-1}, \mathbf{y}_{k-1}, \mathbf{g}_k$, the matrices $\mathbf{S}_{k-1}, \mathbf{Y}_{k-1}, \mathbf{U}_{k-1}, \mathbf{Y}_{k-1}^T\mathbf{Y}_{k-1}, \mathbf{D}_{k-1}$, and the vectors $\mathbf{S}_{k-1}^T\mathbf{g}_{k-1}, \mathbf{Y}_{k-1}^T\mathbf{g}_{k-1}$:

1. Update the $n \times m_{k-1}$ matrices $\mathbf{S}_{k-1}$ and $\mathbf{Y}_{k-1}$ to get the $n \times m_k$ matrices $\mathbf{S}_k$ and $\mathbf{Y}_k$ using $\mathbf{s}_{k-1}$ and $\mathbf{y}_{k-1}$
2. Compute the m_k-vectors $\mathbf{S}_k^T\mathbf{g}_k$ and $\mathbf{Y}_k^T\mathbf{g}_k$.
3. Compute the m_k-vectors $\mathbf{S}_k^T\mathbf{y}_{k-1}$ and $\mathbf{Y}_k^T\mathbf{y}_{k-1}$ by using the fact that

$$\mathbf{y}_{k-1} = \mathbf{g}_k - \mathbf{g}_{k-1}.$$

We already know $m_k - 1$ components of $\mathbf{S}_k\mathbf{g}_{k-1}$ from $\mathbf{S}_{k-1}\mathbf{g}_{k-1}$, and likewise for $\mathbf{Y}_k\mathbf{g}_{k-1}$. We need only compute $\mathbf{s}_{k-1}^T\mathbf{g}_{k-1}$ and $\mathbf{y}_{k-1}^T\mathbf{g}_{k-1}$ and do the subtractions.

No.	Reference	Brief Description
0	§4.4.1	L-BFGS with no options.
1	§4.4.2, Variation 1	Allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}\|\|$ and not allowing m to decrease.
2	§4.4.2, Variation 2	Allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}\|\|$ and allowing m to decrease.
3	§4.4.2, Variation 3	Allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}/\mathbf{x}\|\|$ and not allowing m to decrease.
4	§4.4.2, Variation 4	Allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}/\mathbf{x}\|\|$ and allowing m to decrease.
5	§4.4.3	Dispose of old information if the step length is greater than one.
6	§4.4.4, Variation 1	Back-up if the current iteration is odd.
7	§4.4.4, Variation 2	Back-up if the current iteration is even.
8	§4.4.4, Variation 3	Back-up if a step length of 1.0 was used in the last iteration.
9	§4.4.4, Variation 4	Back-up if $\|\|\mathrm{g}_k\|\| > \|\|\mathrm{g}_{k-1}\|\|$.
10	§4.4.4, Variation 3*	Back-up if a step length of 1.0 was used in the last iteration and we did not back-up on the last iteration.
11	§4.4.4, Variation 4*	Back-up if $\|\|\mathrm{g}_k\|\| > \|\|\mathrm{g}_{k-1}\|\|$ and we did not back-up on the last iteration.
12	§4.4.5, Variation 1	Merge if neither of the two vectors to be merged is itself the result of a merge and the 2nd and 3rd most recent steps taken were of length 1.0.
13	§4.4.5, Variation 2	Merge if we did not do a merge the last iteration and there are at least two old s vectors to merge.
14	§4.4.6, Variation 1	Skip update on odd iterations.
15	§4.4.6, Variation 2	Skip update on even iterations.
16	§4.4.6, Variation 3	Skip update if $\|\|\mathrm{g}_{k+1}\|\| > \|\|\mathrm{g}_k\|\|$.
17	Alg. 5 & Alg. 8	Dispose of old information and back-up on the next iteration if the step length is greater than one.
18	Alg. 13 & Alg. 1	Merge if we did not do a merge the last iteration and there are at least two old s vectors to merge, and allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}\|\|$ and not allowing m to decrease.
19	Alg. 13 & Alg. 3	Merge if we did not do a merge the last iteration and there are at least two old s vectors to merge, and allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}/\mathbf{x}\|\|$ and not allowing m to decrease.
20	Alg. 13 & Alg. 2	Merge if we did not do a merge the last iteration and there are at least two old s vectors to merge, and allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}\|\|$ and allowing m to decrease.
21	Alg. 13 & Alg. 2	Merge if we did not do a merge the last iteration and there are at least two old s vectors to merge, and allow m to vary iteratively basing the choice of m of $\|\|\mathrm{g}/\mathbf{x}\|\|$ and allowing m to decrease.

TABLE 4.2
Description of Numerical Algorithms

4. Compute $\mathbf{U}_k^{-1}$. Rather than recomputing $\mathbf{U}_k^{-1}$, we update the matrix from the previous iteration by deleting the leftmost column and topmost row if $m_k = m_{k-1}$ and appending a new column on the right and a new row on the bottom. Let $\rho_{k-1} = 1/\mathbf{s}_{k-1}^T\mathbf{y}_{k-1}$ and let $(\mathbf{U}_{k-1}^{-1})'$ be the $(m_k-1)\times(m_k-1)$ lower right submatrix of $\mathbf{U}_{k-1}^{-1}$ and let $(\mathbf{S}_k^T\mathbf{y}_{k-1})'$ be the upper $m_k - 1$ elements of $\mathbf{S}_k^T\mathbf{y}_{k-1}$. Then

$$\mathbf{U}_k^{-1} = \begin{pmatrix} (\mathbf{U}_{k-1}^{-1})' & -\rho_{k-1}(\mathbf{U}_{k-1}^{-1})'(\mathbf{S}_k^T\mathbf{y}_{k-1})' \\ 0 & \rho_{k-1} \end{pmatrix}.$$

Note that $\mathbf{s}_{k-1}^T \mathbf{y}_{k-1} = (\mathbf{S}_k^T \mathbf{y}_{k-1})_{m_k}$ and so is already computed.

5. Assemble $\mathbf{Y}_k^T \mathbf{Y}_k$. We have already computed all the components.
6. Update $\mathbf{D}_k$ using $\mathbf{D}_{k-1}$ and $\mathbf{s}_{k-1}^T \mathbf{y}_{k-1} = (\mathbf{S}_k^T \mathbf{y}_{k-1})_{m_k}$.
7. Compute

$$\gamma_k = \mathbf{y}_{k-1}^T \mathbf{s}_{k-1} / \mathbf{y}_{k-1}^T \mathbf{y}_{k-1}.$$

Note that both $\mathbf{y}_{k-1}^T \mathbf{s}_{k-1}$ and $\mathbf{y}_{k-1}^T \mathbf{y}_{k-1}$ have already been computed.

8. Compute two intermediate values

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{U}_k^{-1} \mathbf{S}_k^T \mathbf{g}_k, \\ \mathbf{p}_2 &= \mathbf{U}_k^{-1} (\gamma_k \mathbf{Y}_k^T \mathbf{Y}_k \mathbf{p}_1 + \mathbf{D}_k \mathbf{p}_1 - \gamma_k \mathbf{Y}_k^T \mathbf{g}_k). \end{aligned}$$

9. Compute

$$\mathbf{d}_k = \gamma_k \mathbf{Y}_k \mathbf{p}_1 - \mathbf{S}_k \mathbf{p}_2 - \gamma_k \mathbf{g}_k.$$

The storage costs for this are very low. In order to reconstruct $\mathbf{H}_k$, we need to store $\mathbf{S}_k, \mathbf{Y}_k, \mathbf{U}_k^{-1}, \mathbf{Y}_k^T \mathbf{Y}_k$, $\mathbf{D}_k$ (a diagonal matrix) and a few m-vectors. This requires only $2mn + 2m^2 + O(m)$ storage. Assuming $m << n$, this is much less storage than the n^2 storage required for typical implementation of BFGS.

Step	Operation Count
2	$4mn - 2m$
3	$4n + 2m - 2$
4	$2m^2 - 4m + 3$
7	1
8	$8m^2 + 2m$
9	$4m^2 + 2m$

TABLE 4.3
Operations Count for Computation of $\mathbf{H}_k \mathbf{g}_k$. Steps with no operations are not shown.

The computation of $\mathbf{Hg}$ takes at most $O(mn)$ operations assuming $n >> m$. (See Table 4.3.) This is much less than the $O(n^2)$ time normally needed to compute $\mathbf{Hg}$ when the whole matrix $\mathbf{H}$ is stored.

We are using L-BFGS as our basis for comparison. For information on the performance of L-BFGS see Liu and Nocedal [15] and Nash and Nocedal [19].

4.4.2. Varying m iteratively: Algorithms 1–4. In typical implementations of L-BFGS, m is fixed throughout the iterations: once m updates have accumulated, m updates are always used. We considered the possibility of varying m iteratively, preserving finite termination on convex quadratics. Using an argument similar to that presented in [15], we can also prove that this algorithm has a linear rate of convergence on a convex function that satisfies a few modest conditions.

We tried four different variations on this theme. All were based on the following linear formula that scales m in relation to the size of $\|\mathbf{g}\|$. Let m_k be the number of iterates saved at the kth iteration, with $m_0 = 1$. Here, think of m as the maximum allowable value of m_k. Let the convergence test be given by $\|\mathbf{g}_k\|/\|\mathbf{x}_k\| < \epsilon$. Then the formula for m_k at iteration k is

$$m_k = \min \left\{ m_{k-1} + 1, \left\lfloor (m-1) \frac{\log \delta_k - \log \delta_0}{\log 100\epsilon - \log \delta_0} \right\rfloor + 1 \right\}.$$

Alg. No.	m = 5	m = 10	m = 15	m = 50
0	1	0	0	1
1	0	0	0	0
2	1	0	0	0
3	2	0	0	1
4	1	0	0	1
5	0	0	0	0
6	1	0	0	1
7	0	0	0	1
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	1	0	0	1
13	1	0	0	1
14	11	11	11	11
15	3	3	3	3
16	10	10	8	9
17	0	0	0	0
18	1	1	0	0
19	1	0	0	1
20	1	1	1	1
21	3	1	0	1

TABLE 4.4

The number of failures of the algorithms on the 22 test problems. An algorithm is said to have "failed" on a particular problem if a line search fails or the maximum allowable number of iterations (3000 in our case) is exceeded.

We have two choices for δ_k, and a choice of whether or not we will allow m_k to decrease as well as increase. The four variations are

1. $\delta_k = \|\mathbf{g}_k\|$ and require $m_k \geq m_{k-1}$,
2. $\delta_k = \|\mathbf{g}_k\|$,
3. $\delta_k = \|\mathbf{g}_k\|/\|\mathbf{x}_k\|$ and require $m_k \geq m_{k-1}$, and
4. $\delta_k = \|\mathbf{g}_k\|/\|\mathbf{x}_k\|$.

We used four values of m: 5,10,15 and 50, for each algorithm. The results are summarized in Tables 4.4 – 4.8. More extensive results can be obtained [9].

Table 4.4 shows that these algorithms had roughly the same number of failures as L-BFGS.

Table 4.5 compares each algorithm to L-BFGS in terms of function evaluations. For each algorithm and each value of m, the number of times that the algorithm used *as few or fewer* function evaluations than L-BFGS is listed relative to the total number of admissible problems. Problems are admissible if at least one of the two methods solved it. We observe that in all but three cases, the algorithm used as few or fewer function evaluations than L-BFGS for over half the test problems.

Table 4.6 compares each algorithm to L-BFGS in terms of time. The entries are similar to those in Table 4.5. Observe that Algorithms 1-4 did very well in terms of time, doing as well or better than L-BFGS in nearly every case.

For each problem in each algorithm, we computed the ratio of the number of function evaluations for the algorithm to the number of function evaluations for L-BFGS. Table 4.7 lists the means of these ratios. A mean below 1.0 implies that the algorithm does better than L-BFGS on average. The average is better for the algorithms in 6 out of 16 cases for the first four algorithms. Observe, however, that all the means are close to one.

Alg. No.	m = 5	m = 10	m = 15	m = 50
1	8/22	10/22	17/22	17/22
2	7/22	13/22	13/22	19/22
3	14/21	14/22	12/22	15/21
4	12/21	17/22	15/22	16/21
5	19/22	20/22	20/22	21/22
6	21/21	22/22	22/22	21/21
7	8/22	12/22	10/22	10/22
8	12/22	14/22	12/22	15/22
9	6/22	13/22	12/22	16/22
10	12/22	14/22	12/22	15/22
11	10/22	10/22	11/22	14/22
12	21/21	22/22	22/22	21/21
13	3/22	4/22	4/22	4/22
14	2/21	2/22	2/22	2/21
15	2/22	1/22	1/22	1/22
16	1/22	1/22	1/22	1/22
17	12/22	13/22	12/22	14/22
18	3/22	4/22	5/22	4/22
19	2/22	3/22	4/22	4/22
20	2/22	4/22	4/22	5/22
21	1/22	2/22	4/22	4/22

TABLE 4.5

Function Evaluations Comparison. The first number in each entry is the number of times the algorithm did as well as or better than normal L-BFGS in terms of function evaluations. The second number is the total number of problems solved by at least one of the two methods (the algorithm and/or L-BFGS).

We experience savings in terms of time for the first four algorithms. These algorithms will tend save fewer vectors than L-BFGS since m_k is typically less than m; and so less work is done computing $\mathbf{H}_k\mathbf{g}_k$ in these algorithms. Table 4.8 gives the mean of the ratios of time to solve for each value of m in each algorithm. Note that most of the ratios are far below one in this case.

These variations did particularly well on problem 7. See [9] for more information.

4.4.3. Disposing of old information: Algorithm 5. We may decide that we are storing too much old information and that we should stop using it. For example, we may choose to throw away everything except for the most recent information whenever we take a big step, since the old information may not be relevant to the new neighborhood. We use the following test: If the last step length was bigger than 1, dispose of the old information.

The algorithm performed nearly the same as L-BFGS. There was substantial deviation on only one or two problems for each value of m, and this seemed evenly divided in terms of better and worse. From Table 4.4, we see that this algorithm successfully converged on every problem. Table 4.5 shows that it almost always did as well or better than L-BFGS in terms of function evaluations. However, Table 4.7 shows that the differences were minor. In terms of time, we observe that the algorithm generally was faster than L-BFGS (Table 4.6), but again, considering the mean ratios of time (Table 4.8), the differences were minor. The method also does particularly well on problem 7 [9].

4.4.4. Backing Up in the Update to H: Algorithms 6-11. As discussed in §2.2, if we always use the most recent $\mathbf{s}$ and $\mathbf{y}$ in the update, we preserve quadratic termination regardless of which older values of $\mathbf{s}$ and $\mathbf{y}$ we use.

Alg. No.	m = 5	m = 10	m = 15	m = 50
1	15/22	18/22	20/22	18/22
2	16/22	19/22	18/22	18/22
3	16/21	14/22	15/22	15/21
4	17/21	18/22	20/22	18/21
5	15/22	13/22	14/22	15/22
6	16/21	19/22	15/22	15/21
7	11/22	11/22	10/22	7/22
8	11/22	7/22	6/22	5/22
9	9/22	10/22	7/22	8/22
10	11/22	8/22	5/22	5/22
11	9/22	8/22	9/22	5/22
12	11/21	12/22	8/22	11/21
13	5/22	10/22	13/22	17/22
14	2/21	2/22	2/22	3/21
15	6/22	6/22	9/22	10/22
16	1/22	2/22	3/22	3/22
17	11/22	8/22	5/22	4/22
18	8/22	14/22	19/22	20/22
19	11/22	11/22	17/22	19/22
20	10/22	14/22	17/22	19/22
21	9/22	16/22	16/22	18/22

TABLE 4.6

Time Comparison. The first number in each entry is the number of times the algorithm did as well as or better than normal L-BFGS in terms of time. The second number is the total number of problems solved by at least one of the two methods (the algorithm and/or L-BFGS).

Using this idea, we created some algorithms. Under certain conditions, we discard the next most recent values of **s** and **y** in the **H** although we still use the *most* recent **s** and **y** vectors and any other vectors that have been saved from previous iterations. We call this "backing up" because it as if we back-up over the next most recent values of **s** and **y**. These algorithms used the following four tests to trigger backing up:

1. The current iteration is odd.
2. The current iteration is even.
3. A step length of 1.0 was used in the last iteration.
4. $\|\mathbf{g}_k\| > \|\mathbf{g}_{k-1}\|$.

In two additional algorithms, we varied situations 3 and 4 by not allowing a back-up if a back-up was performed on the previous iteration.

The backing up strategy seemed robust in terms of failures. In 4 out of the 6 variations we did for this algorithm, there were no failures at all. See Table 4.4 for more information.

It is interesting to observe that backing up on odd iterations (Algorithm 6) and backing up on even iterations (Algorithm 7) caused very different results. Backing up on odd iterations seemed to have almost no effect on the number of function evaluations (Table 4.7) and little effect on the time (Table 4.8). However, backing up on even iterations causes much different behavior from L-BFGS. It does worse than L-BFGS on most problems, but better on a few.

Algorithms 8 and 10 were two variations of the same idea: backing up if the previous step length was one. This wipes out the data from the previous iteration after it has been used in one update. Both show improvement over L-BFGS in terms of function evaluations; in fact, these two algorithms have the best function evaluation ratio for the $m = 50$ case (Table 4.7). Unfortunately, these algorithms did not compete with L-BFGS in terms of time (Table 4.8). There is little difference between

Alg. No.	m = 5	m = 10	m = 15	m = 50
1	1.054	1.017	0.931	1.008
2	1.099	0.976	0.968	0.945
3	1.006	0.957	1.391	1.014
4	0.998	1.297	0.970	1.000
5	1.021	0.971	1.005	1.010
6	1.000	1.000	1.000	1.000
7	1.099	0.996	1.205	1.020
8	0.991	1.677	1.507	0.891
9	1.035	1.371	1.005	0.947
10	0.991	1.677	1.507	0.891
11	1.044	0.992	0.981	0.916
12	1.000	1.000	1.000	1.000
13	1.137	1.178	1.244	1.373
14	7.521	7.917	8.288	8.502
15	3.408	4.778	5.292	5.900
16	8.981	5.671	5.807	6.710
17	0.981	1.023	0.924	0.918
18	1.201	1.529	1.209	1.365
19	1.212	1.959	1.242	1.387
20	1.263	1.101	1.226	1.375
21	1.406	1.161	1.178	1.394

TABLE 4.7

Mean function evaluations ratios for each algorithm compared to L-BFGS. Problems for which either method failed are not used in this mean.

Algorithms 8 and 10 — probably because there were rarely many steps of length one is a row.

Algorithms 9 and 11 are also two variations of the same idea: back-up on iteration $k+1$ if the norm of $\mathbf{g}_k$ is bigger than the norm of $\mathbf{g}_{k+1}$. There is a larger difference between the results of 9 and 11 than there was between 8 and 10. In terms of function evaluation ratios (Table 4.7), Algorithm 11 did better, indicating that it may not be wise to back-up twice in a row. Both of these did poorly in terms of time as compared with L-BFGS (Table 4.8).

4.4.5. Merging s and y information in the update: Algorithms 12 and 13. Yet another idea is to "merge" **s** data so that it takes up less storage and computation time. By merging, we mean forming some linear combination of various **s** vectors. The **y** vectors would be merged correspondingly. Corollary 2.5 shows that as long as the most recent **s** and **y** are used without merge, old **s** vectors may be replaced by any linear combination of the old **s** vectors in L-BFGS.

We used this idea in the following way: if certain criteria were met, we replaced the second and third newest **s** vectors in the collection by their sum, and did similarly for the **y** vectors. We used various tests to determine when we would do a merge:

1. Neither of the two vectors to be merged is itself the result of a merge and the second and third most recent steps taken were of length 1.0.

2. We did not do a merge the last iteration and there are at least two old **s** vectors to merge.

The first variation (Algorithm 12) performs almost identically to L-BFGS, especially in terms of time (Table 4.5). Occasionally it did worse in terms of time (Table 4.6). These observations are also reflected in the other results in Table 4.7 and Table 4.8. It is likely that very few vectors were merged.

The second variation (Algorithm 13) makes gains in terms of time, especially for

Alg. No.	m = 5	m = 10	m = 15	m = 50
1	0.972	0.894	0.784	0.884
2	0.993	0.831	0.780	0.783
3	0.955	0.870	1.071	0.898
4	0.907	1.119	0.823	0.856
5	1.041	0.969	0.993	1.004
6	1.007	0.983	0.977	0.995
7	1.088	1.010	1.179	1.692
8	1.057	1.421	1.426	1.425
9	1.032	1.220	1.043	1.173
10	1.056	1.405	1.440	1.412
11	1.062	1.050	1.062	1.208
12	1.008	1.011	1.013	1.002
13	1.083	1.082	0.983	0.960
14	4.585	3.703	3.228	2.417
15	2.318	2.583	2.700	2.633
16	8.589	5.428	4.894	5.956
17	1.053	1.166	1.089	1.399
18	1.081	1.229	0.860	0.885
19	1.130	1.423	0.915	0.923
20	1.114	0.867	0.837	0.916
21	1.258	0.927	0.859	0.974

TABLE 4.8

Mean time ratios for each algorithm compared to L-BFGS. Problems for which either method failed are not used in this mean.

the larger values of m (Table 4.6 and Table 4.8). Unfortunately, this reflects only a saving in the amount of linear algebra required. The number of function evaluations generally is larger for this algorithm than L-BFGS (Table 4.5 and Table 4.7).

4.4.6. Skipping Updates to H: Algorithms 14–16. If every other update to $\mathbf{H}$ is skipped and a step length of one is always chosen, BFGS will terminate in $2n$ iterations on a strictly convex quadratic function. The same holds true when doing an exact line search. (See §3.) Unfortunately, neither property holds in the limited-memory case. We will, however, try some algorithms motivated by this idea.

The idea is that, every so often, we do not use the current $\mathbf{s}$ and $\mathbf{y}$ to update $\mathbf{H}$, and instead just use the old $\mathbf{H}$. There are three variations on this theme.

1. Skip update on odd iterations.
2. Skip update on even iterations.
3. Skip update if $\|\mathbf{g}_{k+1}\| > \|\mathbf{g}_k\|$.

As with the algorithms that did back-ups, the results of the skipping on odd or even iterations were quite different. Skipping on odd updates (Algorithm 14) did extremely well for every value of m on only two problems: 1 and 12. Otherwise, it did very badly. Skipping on even updates (Algorithm 15) performed somewhat better. It did extremely well on problem 7 but not on problems 1 and 12. It also did better than L-BFGS in terms of time on more occasions than Algorithm 14 (Table 4.6). Neither did well in terms of function evaluations, but the mean ratios for function evaluations (Table 4.7) and time (Table 4.8) were usually far greater than one.

Skipping the update if the norm of $\mathbf{g}$ increased (Algorithm 16) did not do well at all. It only did better in terms of function evaluations for one problem for each value of m (Table 4.5) and rarely did better in terms of time (Table 4.6). It ratios were very bad for function evaluations (Table 4.7) and time (Table 4.8)

4.4.7. Combined Methods: Algorithms 17-21. We did some experimentation with combinations of methods described in the previous sections.

In Algorithm 17, we combined Algorithms 5 and 8: we dispose of old information and back-up on the next iterations if the step length is greater than one. Essentially we are assuming that we have stepped out of the region being modeled by the quasi-Newton matrix if we take a long step and we should thus rid the quasi-Newton matrix of that information. This algorithm did well in terms of function evaluations, having mean ratios of less than one for three values of m (Table 4.7), but it did not do as well in terms of time.

In Algorithms 19-21, we combined merging and varying m. These algorithms did well in terms of time for larger m (Table 4.8) but not in terms of function evaluations (Table 4.7).

5. Conclusions. There is a spectrum of quasi-Newton methods, ranging from those that require the storage of an $n \times n$ approximate Hessian (e.g. the Broyden family) to those that require only the storage of a few vectors (e.g. conjugate gradients). Limited-memory quasi-Newton methods fall in between these extremes in terms of performance and storage. There are other methods that fall into the middle ground; for example, conjugate gradient methods such as those proposed by Shanno [27] and Nazareth [20], the truncated-Newton method [24, 6] and the partitioned quasi-Newton method [13].

We have characterized which limited-memory quasi-Newton methods fitting a general form (2.1) have the property of producing conjugate search directions on convex quadratics. We have shown that limited-memory BFGS is the only Broyden family member that has a limited-memory analog with this property. We also considered update-skipping, something that may seem attractive in a parallel environment. We show that update skipping on quadratic problems is acceptable for full-memory Broyden class members in that it only delays termination, but that we lose the property of finite termination if we both limit memory and skip updates.

We have also introduced some simple-to-implement modifications of the standard limited-memory BFGS algorithm that seem to behave well on some practical problems.

Appendix A. Line Search Parameters. Table A.1 give the line search parameters used for our code. Note that in the first iteration, the initial steplength is $\|\mathbf{g}_0\|^{-1}$ rather than 1.0.

Variable	Value	Description
`STP`	1.0	Step length to try first.
`FTOL`	1.0×10^{-4}	Value of ω_1 in Wolfe conditions.
`GTOL`	0.9	Value of ω_2 in Wolfe conditions.
`XTOL`	1.0×10^{-15}	Relative width of interval of uncertainty.
`STPMIN`	1.0×10^{-15}	Minimum step length.
`STPMAX`	1.0×10^{15}	Maximum step length.
`MAXFEV`	20	Maximum number of function evaluations.

TABLE A.1
Line Search Parameters

Appendix B. Pseudo-Code.

B.1. L-BFGS: Algorithm 0. The pseudo-code for the computation of $\mathbf{d}_k = -\mathbf{H}_k \mathbf{g}_k$ at iteration k for L-BFGS is given in Figure B.2. The update of $\mathbf{H}$ is also handled implicitly in this computation.

```
% Compute d_k = -H_k g_k
if (sze == 0)
  d = -g;
else
  % Step 0
  idx = 2 - (sze - oldsze);
  % Step 1
  S = [S(:,idx:oldsze),s];
  Y = [Y(:,idx:oldsze),y];
  % This is needed for Step 3 before we overwrite Stg and Ytg
  Stoldg = [Stg(idx:oldsze); s'*oldg];
  Ytoldg = [Ytg(idx:oldsze); y'*oldg];
  % Step 2
  Stg = S'*g;
  Ytg = Y'*g;
  % Step 3
  Sty = Stg - Stoldg;
  Yty = Ytg - Ytoldg;
  % Step 4
  rho = 1.0/Sty(sze);
  invU = ...
    [invU(idx:oldsze,idx:oldsze) -rho*invU(idx:oldsze,idx:oldsze)*Sty(1:sze-1)
    zeros(1,sze-1) rho];
  % Step 5
  YtY = ...
    [YtY(idx:oldsze,idx:oldsze) Yty(1:sze-1)
    (Yty(1:sze-1))' Yty(sze)];
  % Step 6
  D = [D(idx:oldsze), Sty(sze)];
  % Step 7
  gamma = Sty(sze)/Yty(sze);
  % Step 8
  p1 = invU*Stg;
  p2 = invU*(gamma*YtY*p1 + diag(D)*p1 - gamma*Ytg);
  % Step 9
  d = gamma*Y*p1 - S*p2 - gamma*g;
end
```

FIG. B.1. *MATLAB pseudo-code for the computation of* $\mathbf{d} = \mathbf{Hg}$ *in L-BFGS.* `sze` *is the number of* $\mathbf{s}$ *vectors available for the update this iteration and* `oldsze` *is the number of* $\mathbf{s}$ *vectors that were available the previous iteration. For L-BFGS,* `sze` *is chosen as the minimum of* `oldsze` *+ 1 and* m *(the limited-memory constant).*

B.2. Varying m iteratively: Algorithms 1–4. Suppose that m_k denotes the number of $(\mathbf{s}, \mathbf{y})$ pairs to be used in the kth update. Then simply chose `sze` as the minimum of `oldsze` $+ 1$ and m_k before computing $\mathbf{d}_k$.

B.3. Disposing of old information: Algorithm 5. If the disposal criterion is met at iteration k, set `oldsze` to zero and `sze` to one before computing $\mathbf{d}_k$.

B.4. Backing Up in the Update to H: Algorithms 6-11. If we are to back-up at iterations k, set `oldsze` to the *one less* than the previous value of `sze` and set `sze` as the minimum of `oldsze` $+ 1$ and m, as usual.

B.5. Merging s and y information in the update: Algorithms 12 and 13. Merging is the most complicated variation to handle. Before we determine the newest `sze` and before we compute $\mathbf{d}_k$, we execute the pseudo-code given in Figure B.1. We then set `oldsze` to *one less* than the previous value of `sze` and set `sze` as the minimum of `oldsze` $+ 1$ and m, as usual. We are assuming we are at iteration k, but that the newest values of **s** and **y** have not yet been added to `S` and `Y`.

```
% Execute before choosing new value for sze and before computing d
S(:,sze-1) = S(:,sze) + S(:,sze-1);
Y(:,sze-1) = Y(:,sze) + Y(:,sze-1);
Stg(sze-1) = S(:,sze-1)'*g;
Ytg(sze-1) = Y(:,sze-1)'*g;
delta = S(:,sze-1)'*Y(:,sze-1);
rho = 1.0/delta;
invU = ...
  [invU(1:sze-2,1:sze-2) -rho*invU(1:sze-2,1:sze-2)*S(:,1:sze-2)'*Y(:,sze-1)
  zeros(1,sze-2) rho];
temp = YtY(1:sze-2,sze-1) + YtY(1:sze-2,sze);
YtY = [YtY(1:sze-2,1:sze-2) temp
      temp' Y(:,sze-1)'*Y(:,sze-1)];
D = [D(1:sze-2), delta];
```

FIG. B.2. *MATLAB pseudo-code for the merge variation. This fixes the values of the components that are used in the computation of* $\mathbf{d}_k$.

B.6. Skipping Updates to H: Algorithms 14–16. To skip the update at iteration k, set `sze` to `oldsze`. Compute `Stg` and `Ytg` before Step 0 and then skip to Step 8 and continue.

REFERENCES

[1] I. BONGARTZ, A. R. CONN, N. GOULD, AND P. L. TOINT, `ftp://thales.math.fundp.ac.be/pub/cute`.

[2] ———, `http://www.rl.ac.uk/departments/ccd/numerical/cute/cute.html`.

[3] ———, *CUTE: constrained and unconstrained testing environment*, ACM Transactions on Mathematical Software, 21 (1995), pp. 123–160.

[4] A. BUCKLEY, *Test functions for unconstrained minimization*, Tech. Report TR 1989CS-3, Mathematics, statistics and computing centre, Dalhousie University, Halifax (CDN), 1989. Cited in [1, 2, 3].

[5] A. CONN, N. GOULD, M. LESCRENIER, AND P. TOINT, *Performance of a multifrontal scheme for partially separable optimization*, Tech. Report 88/4, Department of Mathematics, FUNDP, Namur, Belgium, 1988. Cited in [1, 2].

[6] R. S. DEMBO AND T. STEIHAUG, *Truncated-Newton algorithms for large-scale unconstrained optimization*, Mathematical Programming, 26 (1983), pp. 190–212.

[7] J. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Series in Computational Mathematics, Prentice Hall, 1983.

[8] R. FLETCHER, *An optimal positive definite update for sparse Hessian matrices*, Numerical Analysis NA/145, University of Dundee, 1992. Cited in [1, 2].

[9] T. GIBSON, D. O'LEARY, AND L. NAZARETH, `http://www.cs.umd.edu/users/oleary/LBFGS/index.html`, 1996.

[10] P. E. GILL AND W. MURRAY, *Conjugate-gradient methods for large-scale nonlinear optimization*, Tech. Report SOL 79-15, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, 94305, 1979.

[11] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 2nd ed., 1989.

[12] N. GOULD. Private communication to authors of [3]. Cited in [1, 2].

[13] A. GRIEWANK AND P. L. TOINT, *Partitioned variable metric updates for large structured optimization problems*, Numer. Math., 39 (1982), pp. 119–137.

[14] H. KHALFAN, R. BYRD, AND R. SCHNABEL, *A theoretical and experimental study of the symmetric rank one update*, Tech. Report CU-CS-489-90, Department of Computer Science, University of Colorado at Boulder, 1990.

[15] D. C. LIU AND J. NOCEDAL, *On the limited memory BFGS method for large scale optimization*, Mathematical Programming, 45 (1989), pp. 503–528.

[16] D. G. LUENBERGER, *Linear and Nonlinear Programming*, Addison Wesley, 2nd ed., 1984.

[17] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Trans. on Math. Software, 7 (1981), pp. 17–41.

[18] S. Nash, *Newton-type minimization via the Lanczos process*, SIAM Journal on Numerical Analysis, 21 (1984), pp. 770–788.

[19] S. G. Nash and J. Nocedal, *A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization*, SIAM J. Optimization, 1 (1991), pp. 358–372.

[20] L. Nazareth, *A relationship between BFGS and conjugate gradient algorithms and its implications for new algorithms*, SIAM Journal on Numerical Analysis, 16 (1979), pp. 794–800.

[21] ———, *On the BFGS method*. Univ. of California, Berkeley, 1981.

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

[23] ———, *Theory of algorithms for unconstrained optimization*, in Acta Numerica (1991), Cambridge Univ. Press, 1992, pp. 199–242.

[24] D. P. O'Leary, *A discrete Newton algorithm for minimizing a function of many variables*, Mathematical Programming, 23 (1982), pp. 20–33.

[25] S. Oren, *Self-scaling variable metric algorithms, Part II: implementation and experiments*, Management Science, 20 (1974), pp. 863–874. Cited in [1, 2].

[26] M. J. D. Powell, *Quadratic termination properties of minimization algorithms I. Statement and discussion of results.*, J. Inst. Maths Applics, 10 (1972), pp. 333–342.

[27] D. F. Shanno, *Conjugate gradient methods with inexact line searches*, Math. of Oper. Res., 3 (1978), pp. 244–256.

[28] P. Toint, *An error in specifying problem CHNROSNB*. Cited in [1, 2].

[29] ———, *Some numerical results using a sparse matrix updating formula in unconstrained optimization*, Mathematics of Computation, 32 (1978), pp. 839–852.

[30] ———, *Test problems for partially separable optimization and results for the routine PSPMIN*, Tech. Report 83/4, Department of Mathematics, FUNDP, Namur, Belgium, 1983. Cited in [1, 2, 3].