SIAM J. OPTIMIZATION
Vol. 3, No. 3, pp. 582–608, August 1993

009

NUMERICAL EXPERIENCE WITH LIMITED-MEMORY QUASI-NEWTON AND TRUNCATED NEWTON METHODS*

X. ZOU†, I. M. NAVON‡, M. BERGER§, K. H. PHUA¶,
T. SCHLICK‖, AND F. X. LE DIMET**

Abstract. Computational experience with several limited-memory quasi-Newton and truncated Newton methods for unconstrained nonlinear optimization is described. Comparative tests were conducted on a well-known test library [J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *ACM Trans. Math. Software*, 7 (1981), pp. 17–41], on several synthetic problems allowing control of the clustering of eigenvalues in the Hessian spectrum, and on some large-scale problems in oceanography and meteorology. The results indicate that among the tested limited-memory quasi-Newton methods, the L-BFGS method [D. C. Liu and J. Nocedal, *Math. Programming*, 45 (1989), pp. 503–528] has the best overall performance for the problems examined. The numerical performance of two truncated Newton methods, differing in the inner-loop solution for the search vector, is competitive with that of L-BFGS.

Key words. limited-memory quasi-Newton methods, truncated Newton methods, synthetic cluster functions, large-scale unconstrained minimization

AMS subject classifications. 90C30, 93C20, 93C75, 65K10, 76C20

1. Introduction.

Limited-memory quasi-Newton (LMQN) and truncated Newton (TN) methods represent two classes of algorithms that are attractive for large-scale problems because of their modest storage requirements. They use a low and adjustable amount of storage and require the function and gradient values at each iteration. Preconditioning of the Newton equations may be used for both algorithms. In this case, additional function information (e.g., a sparse approximation to the Hessian) may also be required at each iteration. LMQN methods can be viewed as extensions of conjugate-gradient (CG) methods in which the addition of some modest storage serves to accelerate the convergence rate. TN methods attempt to retain the rapid convergence rate of classical Newton methods while economizing storage and computational requirements so as to become feasible for large-scale applications. They can be particularly powerful when structure information of the objective function is exploited [29].

LMQN originated from the works of Nazareth [21] and Perry [25], [26] and were further extended by Shanno [31], [32] resulting in the CONMIN code of Shanno and

*Received by the editors March 29, 1991; accepted for publication (in revised form) April 22, 1992. This work was supported by National Science Foundation grants ATM-8806553, CHE-9002146, and ASC-9157582 (Presidential Young Investigator Award), U.S. Department of Energy contract DE-FC0583ER250000, and the Searle Scholar program.

†Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306.

‡Department of Mathematics and Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306.

§Division of Applied Mathematics, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel.

¶Department of Information and Computer Science, National University of Singapore, Kent Ridge, Singapore 1026, Singapore.

‖Courant Institute of Mathematical Sciences and Department of Chemistry, New York University, 251 Mercer Street, New York, New York 10012.

**Grenoble Laboratoire de Modelization et Calcul, B.P. 53X, 38041 Grenoble Cedex, France.

Phua [33]. Many researchers, including Buckley [1], Nazareth [22], Nocedal [23], Gill and Murray [8], and Nash [15]–[17], studied these methods. Gill and Murray proposed an LMQN method with preconditioning whose code has recently been implemented in routine E04DGF of the NAG library [14]. Buckley and Lenir [3], [4] proposed a variable-storage CG algorithm. The method becomes the usual Shanno–Phua LMQN when the available storage is minimal. Their method was implemented in code BBVSCG, recently updated and improved by Buckley [2]. More recently, the L-BFGS method of Liu and Nocedal [12] based on the limited-memory BFGS method described by Nocedal [23] was developed. L-BFGS is available as routine VA05AD of the Harwell software library. Two TN methods proposed by Nash [15]–[17] and by Schlick and Fogelson [29] have also been made available by the authors for distribution. Here the codes were tested on the variational data assimilation problems in meteorology.

Several large-scale unconstrained minimization algorithms have been previously compared. Navon and Legler [19] compared a number of different CG methods for problems in meteorology and concluded that the Shanno–Phua [33] LMQN algorithm was the most adequate for their test problems. The studies of Gilbert and Lemaréchal [7] and of Liu and Nocedal [12] indicated that the L-BFGS method is among the best LMQN methods available to date. Nash and Nocedal [18] compared the L-BFGS method with the TN method of Nash [15]–[17] on 53 problems of dimensions 10^2 to 10^4. Their results suggested that performance is correlated with the degree of nonlinearity of the objective function: for quadratic and approximately quadratic problems the TN algorithm outperformed L-BFGS, whereas for most of the highly nonlinear problems L-BFGS performed better.

The aim of this paper is to compare and analyze the performance of several LMQN methods. The most representative LMQN method is then compared with TN methods for large-scale problems in meteorology. We focus on various implementation details, such as step-size searches, stopping criteria, and other practical computational features. In §2 we briefly review the tested LMQN methods. The relationships of the different methods to one another are discussed along with practical implementation details. TN methods are briefly described in §3. In §4 we describe the various test problems used in the Moré, Garbow, and Hillstrom [13] package, the synthetic cluster problem, and some real-life large-scale problems ($\sim 10^4$ variables) from oceanography and meteorology. Discussion of the performance of the different LMQN methods and some general observations are presented in §5. In §6 the performance of TN methods for the optimal control problems in meteorology is presented. Summary and conclusions are presented in §7.

2. LMQN algorithms. The behavior of CG algorithms with inexact line searches may depart considerably from theoretical expectations. For this reason, methods such as LMQN compute a descent direction but impose much milder restrictions on the accepted step length.

LMQN algorithms have the following basic structure for minimizing $J(\mathbf{x})$, $\mathbf{x} \in \mathcal{R}^N$:

1) Choose an initial guess $\mathbf{x}_0$ and a positive definite initial approximation to the inverse Hessian matrix $\mathbf{H}_0$ (which may be chosen as the identity matrix).

2) Compute

$$\mathbf{g}_0 = \mathbf{g}(\mathbf{x}_0) = \nabla J(\mathbf{x}_0), \tag{2.1}$$

and set

$$\mathbf{d}_0 = -\mathbf{H}_0\mathbf{g}_0. \tag{2.2}$$

3) For $k = 0, 1, \ldots,$ set

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k, \tag{2.3}$$

where α_k is the step size (see below).

4) Compute

$$\mathbf{g}_{k+1} = \nabla J(\mathbf{x}_{k+1}). \tag{2.4}$$

5) Check for restarts (discussed below).

6) Generate a new search direction $\mathbf{d}_{k+1}$ by setting

$$\mathbf{d}_{k+1} = -\mathbf{H}_{k+1}\mathbf{g}_{k+1}. \tag{2.5}$$

7) Check for convergence: If

$$\|\mathbf{g}_{k+1}\| \leq \epsilon \ \max\{1, \|\mathbf{x}_{k+1}\|\}, \tag{2.6}$$

stop, where $\epsilon = 10^{-5}$. Otherwise, continue from step 3.

LMQN methods combine the advantages of the CG low storage requirement with the computational efficiency of the quasi-Newton (Q-N) method. They avoid storage of the approximate Hessian matrix by building several rank-one or rank-two matrix updates. In practice, the BFGS update formula [12], [24] forms an approximate inverse Hessian from $\mathbf{H}_0$ and k pairs of vectors $(\mathbf{q}_i, \mathbf{p}_i)$, where $\mathbf{q}_i = \mathbf{g}_{i+1} - \mathbf{g}_i$ and $\mathbf{p}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$ for $i \geq 0$. Since $\mathbf{H}_0$ is generally taken to be the identity matrix or some other diagonal matrix, the pairs $(\mathbf{q}_i, \mathbf{p}_i)$ are stored instead of $\mathbf{H}_k$, and $\mathbf{H}_k\mathbf{g}_k$ is computed by a recursive algorithm. All the LMQN methods presented below fit into this conceptual framework. They differ only in the selection of the vector couples $(\mathbf{q}_i, \mathbf{p}_i)$, the choice of $\mathbf{H}_0$, the method for computing $\mathbf{H}_k\mathbf{g}_k$, the line-search implementation, and the handling of restarts.

2.1. CONMIN. The LMQN method of Shanno and Phua [33] is a two-step LMQN-like CG method that incorporates Beale restarts. Only seven vectors of storage are necessary.

Step sizes are obtained by using Davidon's [5] cubic interpolation method to satisfy the following Wolfe [34] conditions:

$$J(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \leq J(\mathbf{x}_k) + \beta' \alpha_k \mathbf{g}_k^T \mathbf{d}_k, \tag{2.7}$$

$$\left| \frac{\nabla J(\mathbf{x}_k + \alpha_k \mathbf{d}_k)^T \mathbf{d}_k}{\mathbf{g}_k^T \mathbf{d}_k} \right| \leq \beta, \tag{2.8}$$

where $\beta' = 0.0001$ and $\beta = 0.9$.

The following restart criterion is used:

$$|\mathbf{g}_{k+1}^T \mathbf{g}_k| \geq 0.2 \|\mathbf{g}_{k+1}\|^2. \tag{2.9}$$

The new search direction $\mathbf{d}_{k+1}$, defined by (2.5), is obtained by setting

$$\mathbf{H}_{k+1} = \hat{\mathbf{H}}_k - \frac{\mathbf{p}_k \mathbf{q}_k^T \hat{\mathbf{H}}_k + \hat{\mathbf{H}}_k \mathbf{q}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k} + \left(1 + \frac{\mathbf{q}_k^T \hat{\mathbf{H}}_k \mathbf{q}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k}. \tag{2.10}$$

If a restart is satisfied, (2.5) is changed to

$$\mathbf{d}_{k+1} = -\hat{\mathbf{H}}_k \mathbf{g}_{k+1}, \tag{2.11}$$

where

$$\hat{\mathbf{H}}_k = \gamma_t \left(\mathbf{I} - \frac{\mathbf{p}_t \mathbf{q}_t^T + \mathbf{q}_t \mathbf{p}_t^T}{\mathbf{p}_t^T \mathbf{q}_t} + \frac{\mathbf{q}_t^T \mathbf{q}_t}{\mathbf{p}_t^T \mathbf{q}_t} \frac{\mathbf{p}_t \mathbf{p}_t^T}{\mathbf{p}_t^T \mathbf{q}_t} \right) + \frac{\mathbf{p}_t \mathbf{p}_t^T}{\mathbf{p}_t^T \mathbf{q}_t}. \tag{2.12}$$

Here the subscript t represents the last step of the previous cycle for which a line search was made. The parameter $\gamma_t = \mathbf{p}_t^T \mathbf{q}_t / \mathbf{q}_t^T \mathbf{q}_t$ is obtained by minimizing the condition number $\mathbf{H}_t^{-1} \mathbf{H}_{t+1}$ [33].

The Shanno and Phua method implemented in CONMIN uses two couples of vectors $\mathbf{q}$ and $\mathbf{p}$ to build its current approximation of the Hessian matrix. The advantage of CONMIN is that it generates descent directions automatically without requiring exact line searches as long as $(\mathbf{q}_k, \mathbf{p}_k)$ are positive at each iteration. This can be ensured by satisfying the second Wolfe condition (2.8) in the line search. However, CONMIN cannot take advantage of additional storage that might be available.

2.2. E04DGF. The Gill and Murray nonlinear unconstrained minimization algorithm is a two-step LMQN method with preconditioning and restarts. The amount of working storage required by this method is $12N$ real words of working space.

The step size is determined as follows. Let $\{\alpha^j, j = 1, 2, \cdots, \}$ define a sequence of points that tend in the limit to a local minimizer of the cost function along the direction $\mathbf{d}_k$. This sequence may be computed by means of a safeguarded polynomial interpolation algorithm. A choice of the initial step length is the one suggested by Davidon [5]:

$$\alpha^0 = \begin{cases} -2(J_k - J_{\text{est}})/\mathbf{g}_k^T \mathbf{d}_k & \text{if } -2(J_k - J_{\text{est}})/\mathbf{g}_k^T \mathbf{d}_k \leq 1, \\ 1 & \text{if } -2(J_k - J_{\text{est}})/\mathbf{g}_k^T \mathbf{d}_k > 1. \end{cases} \tag{2.13}$$

Here J_{est} represents an estimate of the cost function at the solution point. Let t be the first index of this sequence that satisfies

$$|\nabla J(\mathbf{x}_k + \alpha^t \mathbf{d}_k)^T \mathbf{d}_k| \leq -\eta \mathbf{g}_k^T \mathbf{d}_k, \quad 0 \leq \eta \leq 1. \tag{2.14}$$

The method finds the smallest nonnegative integer r such that

$$J_k - J(\mathbf{x}_k + 2^{-r} \alpha^t \mathbf{d}_k) \geq -2^{-r} \alpha^t \mu \mathbf{g}_k^T \mathbf{d}_k, \quad 0 \leq \mu \leq \frac{1}{2}, \tag{2.15}$$

and then sets $\alpha_k = s^{-r} \alpha^t$.

A restart is required if one of the Powell restart criteria (2.9) or the condition

$$-1.2 \|\mathbf{g}_{k+1}\|_2^2 \leq \mathbf{g}_{k+1}^T \mathbf{d}_{k+1} \leq -0.8 \|\mathbf{g}_{k+1}\|_2^2 \tag{2.16}$$

is satisfied [27].

The new search direction is generated by (2.5), where $\mathbf{H}_{k+1}$ is calculated by the following two-step BFGS formula:

$$\mathbf{U}_2 = \mathbf{U}_1 - \frac{1}{\mathbf{q}_k^T \mathbf{p}_k} (\mathbf{U}_1 \mathbf{q}_k \mathbf{p}_k^T + \mathbf{p}_k \mathbf{q}_k^T \mathbf{U}_1) + \frac{1}{\mathbf{q}_k^T \mathbf{p}_k} \left(1 + \frac{\mathbf{q}_k^T \mathbf{U}_1 \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{p}_k} \mathbf{p}_k \mathbf{p}_k^T \right), \tag{2.17}$$

$$\mathbf{H}_{k+1} = \mathbf{U}_2 - \frac{1}{\mathbf{q}_k^T\mathbf{p}_k}(\mathbf{U}_2\mathbf{q}_k\mathbf{p}_k^T + \mathbf{p}_k\mathbf{q}_k^T\mathbf{U}_2) + \frac{1}{\mathbf{q}_k^T\mathbf{p}_k}\left(1 + \frac{\mathbf{q}_k^T\mathbf{U}_2\mathbf{q}_k}{\mathbf{q}_k^T\mathbf{p}_k}\mathbf{p}_k\mathbf{p}_k^T\right). \tag{2.18}$$

If a restart is indicated, the following self-scaling update method [31], [32] is used instead of $\mathbf{U}_2$:

$$\hat{\mathbf{U}}_2 = \gamma\mathbf{U}_1 - \gamma\frac{1}{\mathbf{q}_t^T\mathbf{p}_t}(\mathbf{U}_1\mathbf{q}_t\mathbf{p}_t^T + \mathbf{p}_t\mathbf{q}_t^T\mathbf{U}_1) + \frac{1}{\mathbf{q}_t^T\mathbf{p}_t}\left(1 + \gamma\frac{\mathbf{q}_t^T\mathbf{U}_1\mathbf{q}_t}{\mathbf{q}_t^T\mathbf{p}_t}\mathbf{p}_t\mathbf{p}_t^T\right), \tag{2.19}$$

where $\gamma = \mathbf{q}_t^T\mathbf{p}_t/\mathbf{q}_t^T\mathbf{U}_1\mathbf{q}_t$ and $\mathbf{U}_1$ is a diagonal preconditioning matrix rather than the identity matrix.

2.3. L-BFGS. The LMQN algorithm L-BFGS [12] was chosen as one of the candidate minimization techniques to be tested since it accommodates variable storage, which is crucial in practice for large-scale minimization problems. The method abandons the restart procedure. The update formula generates matrices by using information from the last m Q-N iterations, where m is the number of Q-N updates supplied by the user (generally, $3 \le m \le 7$). After $2Nm$ storage locations are exhausted, the Q-N matrix is updated by replacing the oldest information by the newest information. Thus the Q-N approximation of the inverse Hessian matrix is continuously updated.

In the line search a unit step length is always tried first, and only if it does not satisfy the Wolfe condition is a cubic interpolation performed. This ensures that L-BFGS resembles the (full-memory) BFGS method as much as possible while being as economical as possible for large-scale problems, for which the quadratic termination properties are generally not very meaningful.

$\mathbf{H}_{k+1}$ of (2.5) is obtained by the following procedure. Let $\hat{m} = \min\{k, m-1\}$. Then update $\mathbf{H}_0$ $\hat{m}+1$ times by using the vector pairs $(\mathbf{q}_j, \mathbf{p}_j)_{j=k-\hat{m}}^{k}$, where $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{q}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, and

$$\begin{aligned}\mathbf{H}_{k+1} = {} & (\mathbf{v}_k^T \cdots \mathbf{v}_{k-\hat{m}}^T)\mathbf{H}_0(\mathbf{v}_{k-\hat{m}} \cdots \mathbf{v}_k) \\ & + \rho_{k-\hat{m}}(\mathbf{v}_k^T \cdots \mathbf{v}_{k-\hat{m}+1}^T)\mathbf{p}_{k-\hat{m}}\mathbf{p}_{k-\hat{m}}^T(\mathbf{v}_{k-\hat{m}+1} \cdots \mathbf{v}_k) \\ & + \rho_{k-\hat{m}+1}(\mathbf{v}_k^T \cdots \mathbf{v}_{k-\hat{m}+2}^T)\mathbf{p}_{k-\hat{m}+1}\mathbf{p}_{k-\hat{m}+1}^T(\mathbf{v}_{k-\hat{m}+2} \cdots \mathbf{v}_k) \\ & \vdots \\ & + \rho_k\mathbf{p}_k\mathbf{p}_k^T.\end{aligned} \tag{2.20}$$

Here $\rho_k = 1/(\mathbf{q}_k^T\mathbf{p}_k)$, $\mathbf{v}_k = \mathbf{I} - \rho_k\mathbf{q}_k\mathbf{p}_k^T$, and $\mathbf{I}$ is the identity matrix.

Two options for the above procedure are offered in the code. One performs a more accurate line search by using a small value for β in (2.8) (e.g., $\beta = 10^{-2}$ or $\beta = 10^{-3}$); this is advantageous when the function and gradient evaluations are inexpensive. The other uses simple scaling to reduce the number of iterations. In general it is preferable to replace $\mathbf{H}_0$ of (2.20) by $\mathbf{H}_k^0$ as one proceeds, so that $\mathbf{H}_0$ incorporates more up-to-date information according to one of the following:

M1: $\mathbf{H}_k^0 = \mathbf{H}_0$ (no scaling).

M2: $\mathbf{H}_k^0 = \gamma_0\mathbf{H}_0$, $\gamma_0 = \mathbf{q}_0^T\mathbf{p}_0/\|\mathbf{q}_0\|^2$ (only initial scaling).

M3: $\mathbf{H}_k^0 = \gamma_k\mathbf{H}_0$, $\gamma_k = \mathbf{q}_k^T\mathbf{p}_k/\|\mathbf{q}_k\|^2$.

Since Liu and Nocedal [12] reported that M3 is the most effective scaling, we use it in all our numerical experiments.

2.4. BBVSCG. BBVSCG implements the LMQN method of Buckley and Lenir and may be viewed as an extension of the Shanno and Phua method. Extra storage space can be accommodated.

The method begins by performing the BFGS Q-N update algorithm. When all available storage is exhausted, the current BFGS approximation to the inverse Hessian matrix is retained as a preconditioning matrix. The method then continues by performing preconditioned memoryless Q-N steps, equivalent to the preconditioned CG method with exact line searches. The memoryless Q-N steps are then repeated until the criterion of Powell [27] indicates that a restart is required. At that time all the BFGS corrections are discarded and a new approximation to the preconditioning matrix begins.

For the line search, when $k \leq m$, a step size of $\alpha = 1$ is tried. A line search using cubic interpolation is applied only if the new point does not satisfy $\mathbf{p}_k^T \mathbf{q}_k > 0$. For $k > m$, $\alpha_k = -\mathbf{g}_k^T \mathbf{d}_k / \mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k$. At least one quadratic interpolation is performed before α_k is accepted.

The search direction is calculated by $\mathbf{d}_{k+1} = -\mathbf{H}_k \mathbf{g}_k$ instead of by (2.5), where $\mathbf{H}_k$ is obtained as follows.

(i) If $k = 1$, use a scaled Q-N BFGS formula:

$$\mathbf{H}_1 = \mathbf{\Phi}_0 - \frac{\mathbf{\Phi}_0 \mathbf{q}_k \mathbf{p}_k^T + \mathbf{p}_k \mathbf{q}_k^T \mathbf{\Phi}_0}{\mathbf{p}_k^T \mathbf{q}_k} + \left(1 + \frac{\mathbf{q}_k^T \mathbf{\Phi}_0 \mathbf{q}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k}, \tag{2.21}$$

where $\mathbf{\Phi}_0$ is defined as $\mathbf{\Phi}_0 = (\omega_0 / v_0)\mathbf{H}_0$, $\omega_0 = \mathbf{p}_0^T \mathbf{q}_0$, and $v_0 = \mathbf{q}_0^T \mathbf{H}_0 \mathbf{q}_0$.

(ii) If $1 < k \leq m$, use the Q-N BFGS formula:

$$\mathbf{H}_k = \mathbf{H}_{k-1} - \frac{\mathbf{H}_{k-1} \mathbf{q}_k \mathbf{p}_k^T + \mathbf{p}_k \mathbf{q}_k^T \mathbf{H}_{k-1}}{\mathbf{p}_k^T \mathbf{q}_k} + \left(1 + \frac{\mathbf{q}_k^T \mathbf{H}_{k-1} \mathbf{q}_k}{\mathbf{p}_k^T \mathbf{q}_k}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k^T \mathbf{q}_k}. \tag{2.22}$$

(iii) If $k > m$, use the preconditioned memoryless Q-N formula:

$$\mathbf{H}_k = \mathbf{H}_m - \frac{\mathbf{p}_k \mathbf{q}_k^T \mathbf{H}_m + \mathbf{H}_m \mathbf{q}_k \mathbf{p}_k^T}{\mathbf{p}_k \mathbf{q}_k^T} + \left(1 + \frac{\mathbf{q}_k^T \mathbf{H}_m \mathbf{q}_k}{\mathbf{p}_k \mathbf{q}_k^T}\right) \frac{\mathbf{p}_k \mathbf{p}_k^T}{\mathbf{p}_k \mathbf{q}_k^T}, \tag{2.23}$$

where $\mathbf{H}_m$ is used as a preconditioner.

The matrix $\mathbf{H}_k$ need not be stored since only matrix–vector products $(\mathbf{H}_k \mathbf{v})$ are required. These are calculated from

$$\mathbf{H}_k \mathbf{v} = \mathbf{H}_q \mathbf{v} - \left[\frac{\mathbf{u}_k^T \mathbf{v}}{\omega_k} - \left(1 + \frac{v_k}{\omega_k}\right) \frac{\mathbf{p}_k^T \mathbf{v}}{\omega_k}\right] - \frac{\mathbf{p}_k^T \mathbf{v}}{\omega_k} \mathbf{u}_k, \tag{2.24}$$

where $v_k = \mathbf{q}_k^T \mathbf{H}_q \mathbf{q}_k$, $\omega_k = \mathbf{p}_k^T \mathbf{q}_k$, and $\mathbf{u}_k = \mathbf{H}_q \mathbf{q}_k$. The subscript q is either $k - 1$ or m, depending on whether $k \leq m$ or $k > m$. If one applies (2.24) recursively, the following formula is obtained:

$$\mathbf{H}_q \mathbf{v} = \mathbf{H}_0 \mathbf{v} - \sum_{j=1}^{q} \left\{ \left[\frac{\mathbf{u}_j^T \mathbf{v}}{\omega_j} - \left(1 + \frac{v_j}{\omega_j}\right) \frac{\mathbf{p}_j^T \mathbf{v}}{\omega_j}\right] - \frac{\mathbf{p}_j^T \mathbf{v}}{\omega_j} \mathbf{u}_j \right\}. \tag{2.25}$$

The total storage required for the matrices $\mathbf{H}_1, \ldots, \mathbf{H}_m$ consists of $m(2N + 2)$ locations.

If $k > m$, a restart test is implemented. Restarts will take place if (2.9) and (2.16) are satisfied. In that case $\mathbf{H}_m$ is discarded, k is set to 1, and the algorithm continues from step 1.

Both the L-BFGS and Buckley–Lenir methods allow the user to specify the number of Q-N updates m. When $m = 1$, BBVSCG reduces to CONMIN, whereas when $m = \infty$, both L-BFGS and the Buckley–Lenir methods are identical to the Q-N BFGS method (implemented in the CONMIN-BFGS code).

3. TN methods. Just as LMQN methods attempt to combine modest storage and computational requirements of CG methods with the convergence properties of the standard Q-N methods, TN methods attempt to retain the rapid (quadratic) convergence rate of classic Newton methods while making storage and computational requirements feasible for large-scale applications [6]. Recall that Newton methods for minimizing a multivariate function $J(\mathbf{x}_k)$ are iterative techniques based on minimizing a local quadratic approximation to J at every step. The quadratic model of J at a point $\mathbf{x}_k$ along the direction of a vector $\mathbf{d}_k$ can be written as

$$J(\mathbf{x}_k + \mathbf{d}_k) \approx J(\mathbf{x}_k) + \mathbf{g}_k^T \mathbf{d}_k + \frac{1}{2}\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k, \tag{3.1}$$

where $\mathbf{g}_k$ and $\mathbf{H}_k$ denote the gradient and Hessian, respectively, of J at $\mathbf{x}_k$. Minimization of this quadratic approximation produces a linear system of equations for the search vector $\mathbf{d}_k$ that are known as the Newton equations:

$$\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k. \tag{3.2}$$

In the modified Newton framework a sequence of iterates is generated from $\mathbf{x}_0$ by the rule $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$. The vector $\mathbf{d}_k$ is obtained as the solution (or approximate solution) of the system (3.2) or, possibly, a modified version of it, where some positive definite approximation to $\mathbf{H}_k$, $\tilde{\mathbf{H}}_k$, replaces $\mathbf{H}_k$.

When an approximate solution is used, the method is referred to as a *truncated* Newton method because the solution process of (3.2) is not carried to completion. In this case $\mathbf{d}_k$ may be considered satisfactory when the residual vector $\mathbf{r}_k = \mathbf{H}_k \mathbf{d}_k + \mathbf{g}_k$ is sufficiently small. Truncation may be justified since accurate search directions are not essential in regions far away from local minima. For such regions any descent direction suffices, and so the effort expended in solving the system accurately is often unwarranted. However, as a solution of the optimization problem is approached, the quadratic approximation of (3.1) is likely to become more accurate and a smaller residual may be more important. Thus the truncation criterion should be chosen to enforce a smaller residual systematically as minimization proceeds. One such effective strategy requires

$$\|\mathbf{r}_k\| \leq \eta_k \|\mathbf{g}_k\|, \tag{3.3}$$

where

$$\eta_k = \min\left\{\frac{c}{k}, \|\mathbf{g}_k\|\right\}, \quad c \leq 1. \tag{3.4}$$

Indeed, it can be shown that quadratic convergence can still be maintained [6]. Other truncation criteria have also been discussed [15], [16], [29].

The quadratic subproblem of computing an approximate search direction at each step is accomplished through some iterative scheme. This produces a nested iteration structure: an outer loop for updating $\mathbf{x}_k$ and an inner loop for computing $\mathbf{d}_k$.

The *linear* CG method is attractive for large-scale problems because of its modest computational requirements and theoretical convergence in at most N iterations [9]. However, since CG methods were developed for positive definite systems, adaptations must be made in the present context where the Hessian may be indefinite. Typically, this is handled by terminating the inner loop (at iteration q) when a direction of negative curvature is detected ($\mathbf{d}_q^T \mathbf{H}_k \mathbf{d}_q < \xi$, where ξ is a small positive tolerance such as 10^{-10}); an exit direction that is guaranteed to be a descent direction is then chosen [6], [29]. An alternative procedure to the linear CG for the inner loop is based on the Lanczos factorization [9], which works for symmetric but not necessarily positive definite systems. It is important to note that different procedures for the inner loop can lead to a very different overall performance in the minimization [28].

Implementations of two TN packages are examined in this work: TN1, developed by Nash [15]–[17], which uses a modified Lanczos algorithm with an automatically supplied diagonal preconditioner, and TN2 (TNPACK) developed by Schlick and Fogelson [29] (see also [30]), designed for structured separable problems for which the user provides a sparse preconditioner for the inner loop. In TN2 a sparse modified Cholesky factorization based on the Yale Sparse Matrix Package is used to factor the preconditioner, which need not be positive definite (computational chemistry problems, where such situations occur, provided motivation for the method). Two modified Cholesky factorizations have been implemented in TN2 [28]. Although we have not yet formulated a preconditioner for our meteorology application, we intend to focus future efforts on formulating an efficient preconditioner for this package. Here we report only results for which no preconditioning is used in TN2. Although it is clear that performance must suffer, our results provide further perspective. Full algorithmic descriptions of the TN codes can be found in the original cited works.

4. Testing problems. Moré, Garbow, and Hillstrom [13] developed a relatively large collection of carefully coded test functions of different degrees of difficulty and designed very simple procedures for testing the reliability and robustness of the optimization software. We used these problems to test the different LMQN methods.

The test problems of [13] involve Hessians of varying spectral condition numbers and eigenvalues, and the eigenvalues are generally of unknown and uncontrollable dispersion. A synthetic test function with a controllable spectrum of clustered eigenvalues was thus also tested.

Two representative real-life large-scale unconstrained minimization applications from meteorology and oceanography were also examined to compare the performances of the LMQN and TN methods. The number of variables for these large-scale problems ranges from 7330 to 14,763.

4.1. Standard library test problems. All of the 18 test problems of Moré, Garbow, and Hillstrom for unconstrained minimization have the following composition:

$$J(\mathbf{x}) = \sum_{i=1}^{m} f_i^2(\mathbf{x}), \quad m \leq N, \quad \mathbf{x} \in \mathcal{R}^N. \tag{4.1}$$

These problems were all minimized by using both the recommended standard starting points $\mathbf{x}_0$ as well as by using nonstandard starting points, taken as $10\mathbf{x}_0$ and $100\mathbf{x}_0$. The vectors $\mathbf{x}_0$ and $100\mathbf{x}_0$ are regarded as being close to and far away from the solution, respectively; it is not unusual for unconstrained minimization algorithms to succeed with an initial guess of $\mathbf{x}_0$ but fail with an initial guess of either $10\mathbf{x}_0$ or $100\mathbf{x}_0$.

4.2. Synthetic cluster function problems. Consider the quadratic objective function

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}, \tag{4.2}$$

where $\mathbf{x}$ is a vector of N variables and $\mathbf{H}$ is an $N \times N$ positive definite matrix of real entries. There exists then a real orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{Q}^T\mathbf{H}\mathbf{Q} = \text{diag}(\lambda_1, \ldots, \lambda_N), \tag{4.3}$$

where the ith column of $\mathbf{Q}$ is the ith eigenvector corresponding to the ith eigenvalue λ_i. The objective function can be written as

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{D}\mathbf{Q}^T\mathbf{x}. \tag{4.4}$$

The orientation and shape of this N-dimensional quadratic surface is a function of $\mathbf{Q}$ and $\mathbf{D}$: the directions of the principal axes of this hyperellipsoid are determined by the directions of the eigenvectors, and the lengths of the axes are determined by the eigenvalues. The axes' lengths are inversely proportional to the square root of the corresponding eigenvalues.

Consider a quadratic objective function defined by

$$J(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{N}(D_{ii}x_i)^2, \tag{4.5}$$

where

$$D_{ii} = \left(1 + \frac{i - M_{k-1} - \lfloor \frac{N_k}{2} \rfloor - 1}{\lfloor \frac{N_k}{2} \rfloor + 1} D_k\right) c_k, \tag{4.6}$$

$\lfloor\ \rfloor$ represents the floor function, N_k, M_k, and K are some positive integers, and c_k and D_k are some real values satisfying the following restrictions:

$$\sum_{k=1}^{K} N_k = N, \quad 1 \le K \le N,$$

$$c_1 < c_2 < \cdots < c_K, \quad 0 \le D_k < 1, \tag{4.7}$$

$$M_k = \sum_{m=1}^{k} N_m, \quad M_0 = 0.$$

By comparing (4.7) with (4.4) we see that the function (4.5) has the standard basis eigenvectors (since $\mathbf{Q}$ in this case was taken to be equal to the identity matrix $\mathbf{I}$) and K clusters of eigenvalues with N_k eigenvalues in the kth cluster, respectively. The kth cluster is located around the position c_k with interval width D_k, which is defined in a fractional form in terms of c_k ($0.0 \le D_k < 1.0$). For example, $D_k = 0.5$ implies an interval width of $[0.5c_k, 1.5c_k]$.

This function yields an eigenvalue system of homogeneous dispersion within each cluster, i.e., each cluster consists of equally spaced eigenvalues. The choice $\mathbf{Q} = \mathbf{I}$ determines an orientation of the hyperellipsoid in which the principal axes are aligned parallel to the $\mathbf{x}$ coordinates. The advantage of this choice is that, without loss of generality, the objective function and its gradient vector are computationally very simple even for large N, which permits testing on very large problems at a relatively low cost. The gradient components for this choice are given by

$$G_i = (D_{ii})^2 x_i, \quad i = 1, \ldots, N, \tag{4.8}$$

and the condition number of the Hessian is given by

$$c_n = \left(\frac{D_{NN}}{D_{11}}\right)^2. \tag{4.9}$$

As shown below, we can test the LMQN methods with a variety of setup values for the various parameters $(N; K; N_k, k = 1, \ldots, K; c_k, k = 1, \ldots, K; D_k, k = 1, \ldots, K)$.

Example 1. Let $N = 21; K = 1; N_1 = N; D_1 = A; C_1 = 1.0$. These parameters yield N-dimensional hyperellipsoidal contours. The condition number of this system is $c_n = ((11 + 10A)/(11 - 10A))^2$.

Example 2. Let $N = 21; K = 2, N_1 = 11, N_2 = 10; C_1 = 1.0, C_2 = A; D_1 = 0.2, D_2 = 0.3$. These parameters yield a bicluster problem, the condition number being controlled by $c_n = ((6 + 4D_2)c_2/(6 - 5D_1)c_1)^2$.

4.3. Oceanography problem. This problem is derived from an analysis of the monthly averaged pseudo-wind-stress components over the Indian Ocean. We attempt to analyze the wind over a region Ω by using the following available information: (a) ship-reported averages on a 1° resolution mesh and (b) a 60-yr pseudostress climatology. The objective function is a measure of discrepancy in the data according to certain prescribed conditions, which may be dynamically or statistically motivated. According to climatological observations, the wind pattern should be smooth. Some measure of roughness and some measure of lack of fit to climatology should also be included in the objective function [11].

To formulate the problem, we used the following objective function:

$$\begin{aligned} J(\tau_x, \tau_y) = {} & \frac{1}{L^2} \sum_x \sum_y [(\tau_x - \tau_{x_0})^2 + (\tau_y - \tau_{y_0})^2] \\ & + \frac{\gamma}{L^2} \sum_x \sum_y [(\tau_x - \tau_{x_c})^2 + (\tau_y - \tau_{y_c})^2] \\ & + L^2 \lambda \sum_x \sum_y \left[(\nabla^2(\tau_x - \tau_{x_c}))^2 + (\nabla^2(\tau_y - \tau_{y_c}))^2\right] \\ & + \beta \sum_x \sum_y [\nabla \cdot (\tau_{\mathbf{x}} - \tau_{\mathbf{x_c}})]^2 + \alpha \sum_x \sum_y [\mathbf{k} \cdot \nabla \times (\tau_{\mathbf{x}} - \tau_{\mathbf{x_c}})]^2, \end{aligned} \tag{4.10}$$

where τ_{x_0} and τ_{y_0} are the components of the 1° mean values determined by the ship wind reports; τ_{x_c} and τ_{y_c} are climatology pseudostress vectors, respectively; $\tau_x = u \cdot (u^2 + v^2)^{1/2}$ and $\tau_y = v \cdot (u^2 + v^2)^{1/2}$ are the resultant eastward and northward pseudostress components, respectively; $\mathbf{v}$ represents the wind vector; and L is a length scale (chosen to be 1° latitude), which makes all the terms in the objective cost

function dimensionally uniform and scales them to the same order of magnitude. The coefficients (actually weights) γ, λ, β, and α control how closely the direct minimization fits each constraint. The first term in J expresses the closeness to the input data. The second measures the fit to the climatology data values for that month. The third is a smoothing term for data roughness and controls the radius of influence of an anomaly in the input data. The fourth and the fifth terms are boundary-layer kinematic terms that force the results to be comparable to the climatology.

A discretization of the domain Ω of 3665 mesh points produces $2 \times 3665 = 7330$ variables.

4.4. Meteorology problems. Combining in an optimal way new information (in the form of measurements) with a priori knowledge (in the form of a forecast) is a key idea of variational data assimilation in numerical weather prediction. The object is to produce a regular, physically consistent two- or three-dimensional representation of the state of the atmosphere from a series of measurements (called observations) that are heterogeneous in both space and time. This approach is implemented by minimizing a cost function measuring the misfit between the observations and the solution predicted by the model.

Below, a two-dimensional limited-area shallow-water-equations model is used to evaluate a quadratic objective function. The equations may be written as the following (see [20] for details):

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - fv + \frac{\partial \phi}{\partial x} = 0, \tag{4.11a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + fu + \frac{\partial \phi}{\partial y} = 0, \tag{4.11b}$$

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y} + \phi\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0, \tag{4.11c}$$

where f is the Coriolis parameter, u, v are the two components of the velocity field, and ϕ is the geopotential field; both fields are spatially discretized with a centered-difference scheme in space and an explicit leap-frog integration scheme in time. A rectangular domain of size $L = 6000\,\text{km}$, $D = 4400\,\text{km}$ is used along with discretization parameters $\Delta x = 300\,\text{km}$, $\Delta y = 220\,\text{km}$, and $\Delta t = 600\,\text{s}$.

This model is widely used in meteorology and oceanography since it contains most of the physical degrees of freedom (including gravity waves) present in the more sophisticated three-dimensional primitive-equation models. It is computationally less expensive to implement, and results with this model can be expected to be similar to those obtained from a more complicated primitive-equation model. The gradient of the objective function with respect to the control variables is calculated by the adjoint technique [20].

Two experiments are conducted here. The first involves a model in which only the initial conditions serve as the control variables. The second includes both the initial and boundary conditions as control variables.

The objective function is defined as a simple weighted sum of squared differences between the observations and the corresponding prediction model values:

$$J = W_\phi \sum_{n=1}^{N_\phi} (\phi_n - \phi_n^{\text{obs}})^2 + W_V \sum_{n=1}^{N_V} [(u_n - u_n^{\text{obs}})^2 + (v_n - v_n^{\text{obs}})^2], \tag{4.12}$$

where u, v are the two components of the velocity field, ϕ is the geopotential field, N_ϕ is the total number of geopotential observations available over the assimilation window (t_0, t_R), and N_V is the total number of wind vector observations. The quantities $u_n^{\text{obs}}, v_n^{\text{obs}}$, and ϕ_n^{obs} are the observed values for the northward wind component, the eastward wind component, and the geopotential field, respectively, and the quantities u_n, v_n, and ϕ_n are the corresponding computed model values. W_ϕ and W_V are weighting factors, taken to be the inverse of estimates of the statistical root-mean-square observational errors on geopotential and wind components, respectively. Values of $W_\phi = 10^{-4}\,\text{m}^{-4}\text{s}^4$ and $W_V = 10^{-2}\,\text{m}^{-2}\text{s}^2$ are used. In the first problem the objective function J is viewed as a function of $\mathbf{x}_0 = (u(t_0), v(t_0), \phi(t_0))^T$, whereas in the second J is a function of $(\mathbf{x}_0, \mathbf{v})$, where $\mathbf{v}$ represents a function of time defined on the boundary.

For the experiments the observational data consist of the model-integrated values for wind and geopotential at each time step starting from the Grammeltvedt initial conditions [10] (see Fig. 1). Random perturbations of these fields, performed by using a standard library randomizer RANF on the CRAY-YMP (shown in Fig. 2), are then used as the initial guess for the solution. A grid of 21×21 points in space and 60 time steps in the assimilation window (10 hrs) results in a dimension of the vector of control variables of 1323 for the initial control problem. Controlling the boundary conditions of a limited-area model implies storing in memory as control variables all of the three field variables on the boundary perimeter for all the time steps. The dimension of the vector of control variables thus becomes 14,763.

Two different scaling procedures were considered: gradient and consistent. The first scales the gradient of the objective function. The second makes the shallow-water-equations model nondimensional.

5. Numerical results for LMQN methods. In most of our test problems (those in [13] and synthetic problems) the computational cost of the function is low and the computational effort of the minimization iteration sometimes dominates the cost of evaluating the function and gradient. However, there are also several practical large-scale problems (for example, the variational data assimilation in meteorology) for which the functional computation is expensive. We report, therefore, both the number of function and gradient evaluations and the time required for minimization of some problems.

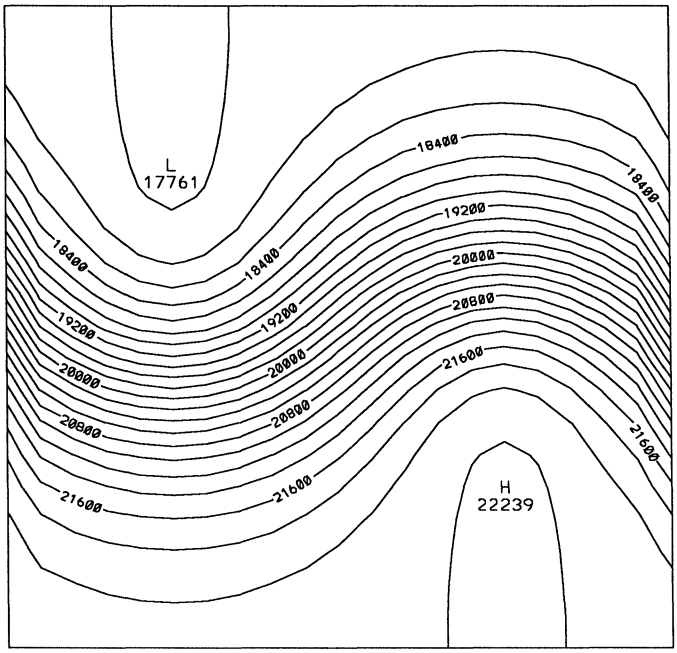
Table 1 shows the amount of storage required by the different LMQN methods for various values of m, the number of Q-N updates, and the dimension N.

The runs below were performed on a CRAY-YMP, for which the unit roundoff is approximately 10^{-14}. In all tables "Iter" represents the total number of iterations, "Nfun" represents the total number of function calls, "MTM" represents the total CPU time spent in minimization, and "FTM" represents the CPU time spent in function and gradient evaluations.

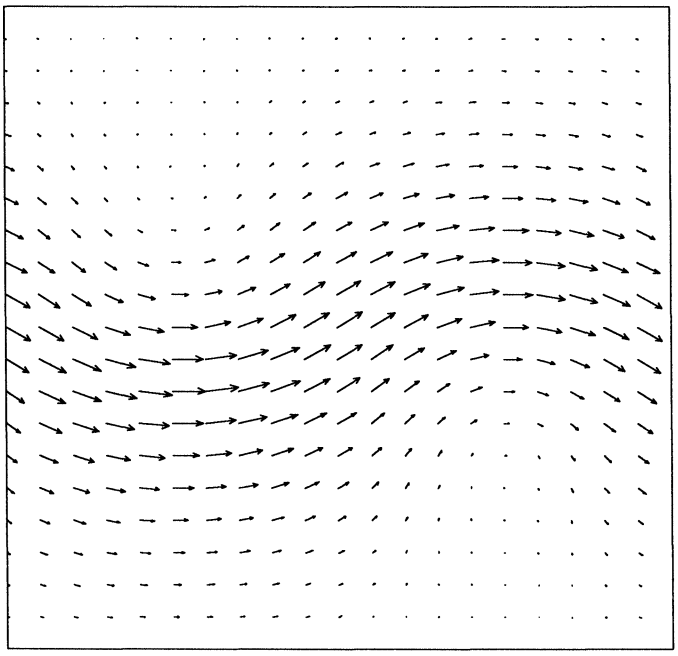
5.1. Results for the standard library test problems. For the 18 test problems, the number of variables ranges from 2 to 100. All the runs reported in this section and §5.2 were terminated when the stopping criterion (2.6) was satisfied. Low accuracy in the solution is adequate in practice.

In the corresponding tables P denotes the problem number, and the results are reported in the form

CONMIN-CG/CONMIN-BFGS/E04DGF,
L-BFGS ($m = 3$)/L-BFGS ($m = 5$)/L-BFGS ($m = 7$),
BBVSCG ($m = 3$)/BBVSCG ($m = 5$)/BBVSCG ($m = 7$).

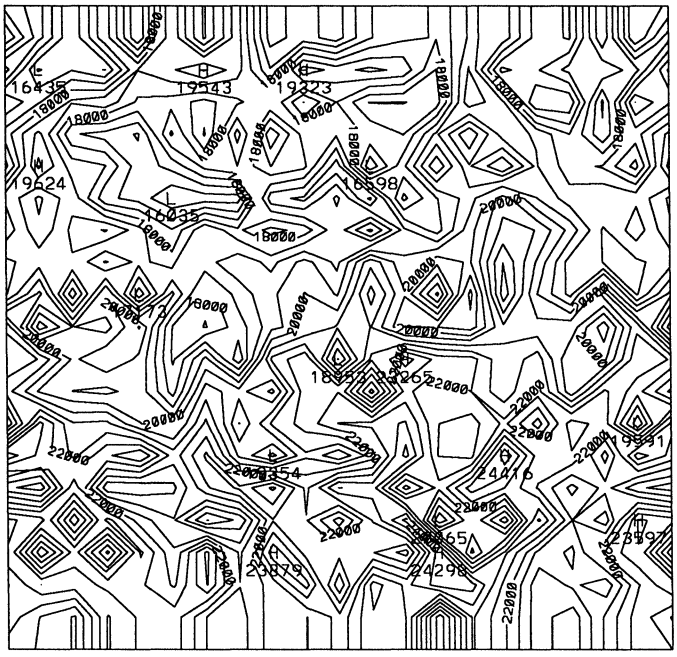


(a)

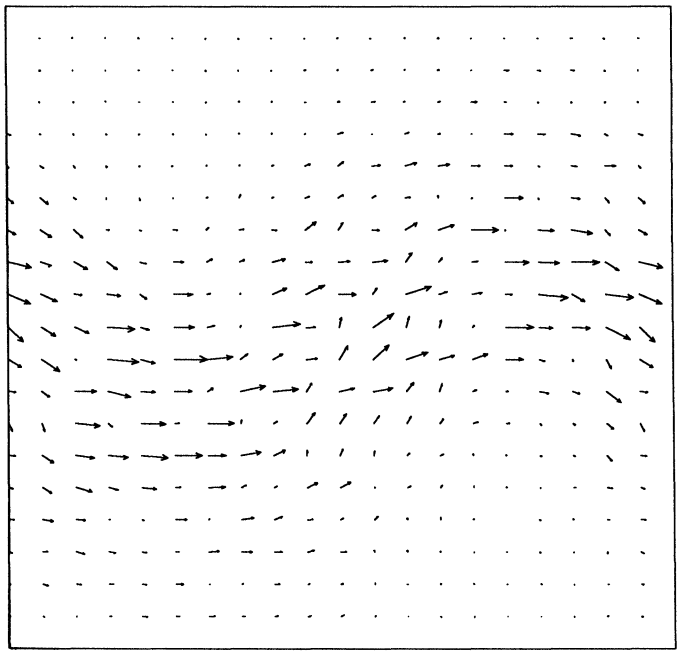


(b)

FIG. 1 *Geopotential field* (a) *based on the Grammeltvedt initial condition and the wind field* (b), *calculated from the geopotential fields in Fig.* 1(a) *by the geostrophic approximation at the same time levels. Contour interval is* 200 m^2s^{-2} *and the value of maximum vector is* 29.9 ms^{-1}.



(a)



(b)

FIG. 2 *Random perturbation of the geopotential* (a) *and the wind* (b), *fields in Fig.* 1. *Contour interval is* 500 m^2s^{-2} *and the value of maximum vector is* 54.4 ms^{-1}.

TABLE 1

Storage locations (N, dimension of the control variable; m, number of quasi-Newton updates).

CONMIN-CG	CONMIN-BFGS	E04DGF	L-BFGS	BBVSCG
5N+2	N(N+7)/2	14N	(2m+2)N+2m	(2m+3)N+2m

Table 2 compares the performance of the four LMQN methods from the standard starting points, with $m = 3, 5, 7$ updates for both L-BFGS and the Buckley–Lenir method. An "F" indicates failure when the maximum number of function calls (3000) is exceeded. An "S" indicates failure in the line search. The latter may occur from roundoff, and a solution may be obtained nonetheless.

The results show that for some problems in which the objective function depends on no more than three or four variables (such as problems 4, 10, 12, and 16) the full-memory Q-N BFGS method is clearly superior to the LMQN methods. For other problems the LMQN methods display better performance.

For most problems the number of iterations and function calls required decreases as the number of Q-N updates m is increased in L-BFGS. A dramatic case illustrating this is the extended Powell singular function (problem 15). The variation of the value of the objective function and the norm of the gradient with the number of iterations is shown in Fig. 3. For $m = 3$ the number of iterations and function calls required to reach the same convergence criteria is (65, 76); for $m = 5$ it is (56, 66), and for $m = 7$ it is (39, 45). The difference between different values of m becomes obvious only after 18 iterations.

BBVSCG usually uses the fewest function calls when $m = 7$. Either $m = 7$ or $m = 3$ performs best in terms of the number of iterations. Figures 4 and 5 present two illustrative examples. Figure 4 presents the variation of the value of J and the norm of ∇J for the Wood function (problem 17), and Fig. 5 presents the same variation for the variable-dimensioned function (problem 6 ($N = 100$)). The differences between the cases $m = 5$ and $m = 7$ for the two problems are smaller than the corresponding differences between the $m = 3$ and $m = 5$ cases.

Table 2 also shows that L-BFGS usually requires fewer function calls than does BBVSCG. This agrees with the experience of Liu and Nocedal [12], who suggested that BBVSCG gives little or no "speed-up" from additional storage. To investigate this further, we measure in Figs. 6 and 7 the effect of increasing the storage. We define the speed-up by using the same definition as did Liu and Nocedal, i.e., the ratio of function calls required when $m = 3$ and $m = 7$.

We see from these figures that the speed-up of BBVSCG is not smaller than that of L-BFGS. There are cases for which L-BFGS gains more speed-up than does BBVSCG (i.e., problems 2, 4, 5, 7a, 9b, 11, 15, 18). However, there are also cases for which BBVSCG has larger speed-up than does L-BFGS (i.e., problems 7b, 8, 9a, 12, 13, 16, 17). Therefore, the reason that L-BFGS requires fewer function calls cannot be the difference in speed-up between the two codes.

For problems for which the function and gradient evaluations are inexpensive, we also examine the number of iterations and the total time required by the two methods. From Table 2 we see that BBVSCG usually requires fewer iterations and less total CPU time than does L-BFGS. The more accurate line search in BBVSCG may provide an explanation. Will a more accurate line search in L-BFGS decrease the number of iterations? In Table 3 we present the results for L-BFGS ($m = 7$) when the line search is forced to satisfy (2.8) with $\beta = 0.01$ rather than 0.9.

For most problems (18 out of 21) the number of iterations when L-BFGS is used

TABLE 2

Eighteen standard library test problems with standard starting points.

P	N	Iter	Nfun	MTM (total CPU time)	FTM (function calls' CPU)
1	3	22/28/37	49/31/81	0.0223/0.0238/0.0211	0.0008/0.0005/0.0014
		28/27/28	35/31/34	0.0278/0.0316/0.0383	0.0006/0.0005/0.0006
		25/31/26	42/44/39	0.0229/0.0326/0.0315	0.0007/0.0007/0.0007
2	6	24/41/48	50/45/100	0.0260/0.0557/0.0301	0.0026/0.0023/0.0051
		52/42/35	66/46/42	0.0556/0.0523/0.0513	0.0034/0.0024/0.0021
		45/52/38	80/86/55	0.0456/0.0619/0.0516	0.0041/0.0044/0.0029
3	3	3/7/4	7/9/11	0.0028/0.0058/0.0070	0.0001/0.0002/0.0002
		6/7/7	9/9/9	0.0059/0.0073/0.0074	0.0002/0.0002/0.0002
		4/5/4	9/9/8	0.0032/0.0042/0.0031	0.0002/0.0002/0.0001
4	2	99/140/167	257/187/433	0.0912/0.1042/0.0770	0.0033/0.0024/0.0051
		173/167/169	229/208/215	0.1752/0.2045/0.2461	0.0030/0.0027/0.0028
		114/123/136	232/235/231	0.1066/0.1359/0.1723	0.0030/0.0031/0.0030
5	3	11/32/47	31/40/113	0.0109/0.0285/0.0278	0.0010/0.0013/0.0037
		30/29/27	40/40/37	0.0310/0.0361/0.0383	0.0013/0.0013/0.0012
		19/22/22	36/35/36	0.0175/0.0230/0.0279	0.0012/0.0011/0.0012
6	10	6/19/19	13/20/41	0.0052/0.0373/0.0148	0.0002/0.0003/0.0006
		19/19/19	20/20/20	0.0187/0.0224/0.0256	0.0003/0.0003/0.0003
		14/13/17	26/22/27	0.0129/0.0142/0.0218	0.0004/0.0003/0.0004
	100	10/36/35	21/37/73	0.0224/1.7774/0.0464	0.0004/0.0007/0.0014
		36/36/36	37/37/37	0.0733/0.0936/0.1125	0.0007/0.0007/0.0007
		15/26/29	37/49/45	0.0288/0.0586/0.0707	0.0007/0.0010/0.0009
7	12	215/97/F	432/100/F	0.4707/0.2870/2.1152	0.2152/0.0498/1.4839
		2202/396/222	2511/458/255	3.5291/0.7419/0.4794	1.2506/0.2282/0.1276
		239/234/201	433/373/289	0.4597/0.4616/0.4111	0.2158/0.1859/0.1440
	30	600/135/F	1208/140/F	2.1548/1.1544/3.8358	1.2577/0.1457/3.1262
		F/2049/106	F/2400/1240	6.2462/5.7458/3.2839	3.1237/2.4986/1.2903
		572/470/223	1108/800/355	1.8518/1.4923/0.7219	1.1542/0.8334/0.3697
8	100	24/71/F	59/80/F	0.0604/3.5806/0.7649	0.0010/0.0014/0.0512
		60/59/56	73/69/67	0.1266/0.1590/0.1824	0.0012/0.0012/0.0011
		67/48/49	132/87/80	0.1276/0.1066/0.1271	0.0023/0.0015/0.0014
9	10	125/17/332	306/26/863	0.1437/0.0355/0.1997	0.0173/0.0015/0.0484
		17/21/17	19/23/19	0.0182/0.0261/0.0236	0.0011/0.0013/0.0011
		18/17/18	32/25/24	0.0186/0.0190/0.0228	0.0018/0.0014/0.0014
	50	127/F/141	281/F/309	0.2719/9.3038/0.1596	0.0650/0.6949/0.0712
		250/250/223	331/308/272	0.4425/0.5301/0.5577	0.0766/0.0712/0.0628
		136/146/148	254/256/215	0.2381/0.2883/0.3168	0.0589/0.0593/0.0498
10	2	8/10/11	18/15/28	0.0063/0.0073/0.0097	0.0001/0.0001/0.0002
		13/13/13	25/25/25	0.0142/0.0162/0.0177	0.0002/0.0002/0.0002
		11/11/16	26/22/30	0.0098/0.0108/0.0191	0.0002/0.0002/0.0002
11	4	26/35/36	F/36/79	0.4817/0.0356/0.0231	0.1299/0.0016/0.0034
		38/22/27	63/43/37	0.0432/0.0305/0.0387	0.0027/0.0019/0.0016
		25/24/23	S/48/52	0.0344/0.0262/0.0279	0.0051/0.0016/0.0014
12	3	15/16/29	36/21/71	0.0207/0.0183/0.0330	0.0081/0.0048/0.0161
		26/24/24	35/32/33	0.0337/0.0355/0.0402	0.0079/0.0072/0.0075
		13/13/15	30/27/24	0.0187/0.0199/0.0222	0.0068/0.0061/0.0054
13	100	46/42/53	98/52/122	0.1457/2.1017/0.0749	0.0142/0.0075/0.0176
		49/48/47	56/54/53	0.1094/0.1345/0.1574	0.0081/0.0078/0.0077
		43/54/51	78/91/48	0.0968/0.1431/0.1422	0.0113/0.0132/0.0107
14	100	18/36/24	49/50/69	0.04571.7946/0.0378	0.0006/0.0006/0.0009
		33/35/38	45/46/50	0.0706/0.0941/0.1227	0.0005/0.0006/0.0006
		31/30/34	56/48/52	0.0576/0.0645/0.0854	0.0007/0.0006/0.0006
15	100	47/42/46	95/43/108	0.1247/2.0963/0.0536	0.0014/0.0006/0.0016
		65/56/39	76/66/45	0.1367/0.1512/0.1243	0.0011/0.0010/0.0007
		44/49/58	86/79/82	0.0854/0.1087/0.1494	0.0013/0.0012/0.0012
16	2	10/15/16	22/16/35	0.0078/0.0103/0.0114	0.0002/0.0001/0.0003
		16/14/14	18/15/15	0.0150/0.0151/0.0169	0.0001/0.0001/0.0001
		14/16/15	30/25/19	0.0130/0.0161/0.0163	0.0002/0.0002/0.0002
17	4	48/36/200	106/43/418	0.0497/0.0362/0.0871	0.0009/0.0004/0.0035
		106/93/86	137/119/113	0.1073/0.1152/0.1260	0.0011/0.00100.0009
		30/24/22	53/42/34	0.0272/0.0253/0.0262	0.0004/0.0003/0.0003
18	100	686/465/832	1384/491/1724	8.8249/26.57/9.0296	6.7316/2.39398.4209
		1137/853/781	1213/895/821	8.2528/6.6732/6.6005	5.8978/4.3587/3.9912
		709/591/698	1393/1110/1318	8.5565/7.2290/9.2365	6.7916/5.4006/6.5045

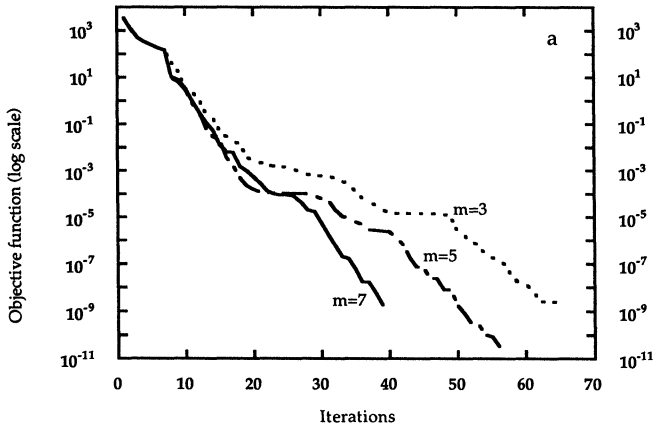
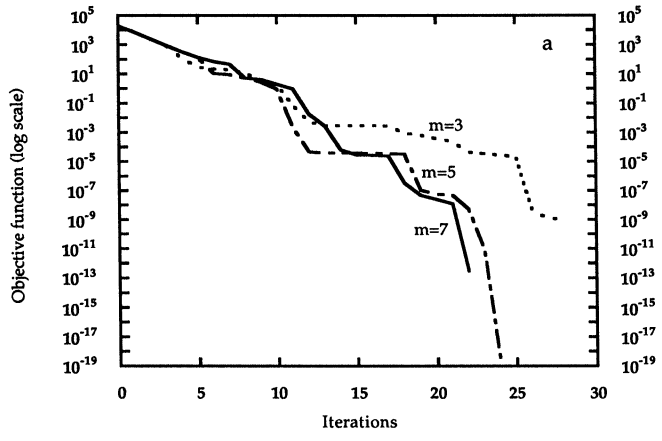


FIG. 3. *Variation of* (a) *the objective function and* (b) *the norm of gradient with the number of iterations using the* L-BFGS *method with m equal* 7 (*solid*), 5 (*dash dot*), *and* 3 (*dotted*) *for the test library problem* 15.

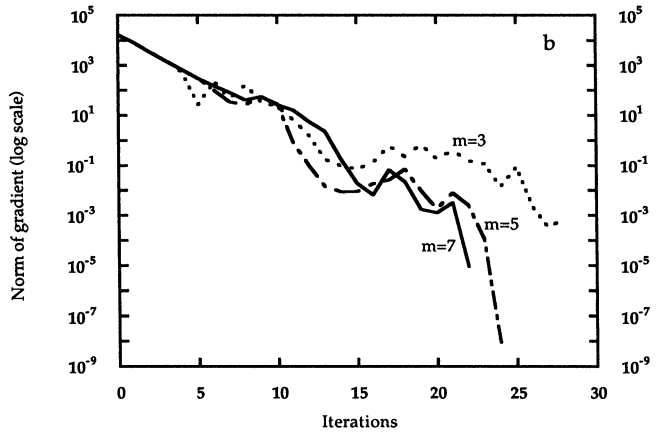
is then markedly reduced (compare Table 2 L-BFGS ($m = 7$) with Table 3). Among those problems, about two-thirds require more function calls, but about one-third require even fewer function calls.

This implementation of L-BFGS is compared with the CONMIN-CG, E04DGF, L-BFGS ($\beta = 0.9$), and BBVSCG codes in Table 4. The "number of wins" describes the number of runs for which a method required fewest function calls and the number of runs for which a method required fewest iterations. Because ties occur, numbers across a row do not add up to the number of different test cases.

We see that L-BFGS ($m = 7$ and $\beta = 0.01$) uses the fewest iterations and that L-BFGS ($m = 7$ and $\beta = 0.9$), CONMIN-CG, and BBVSCG use the fewest function calls. If both the numbers of iterations and function calls are considered, CONMIN-CG seems to be the best.



(a)



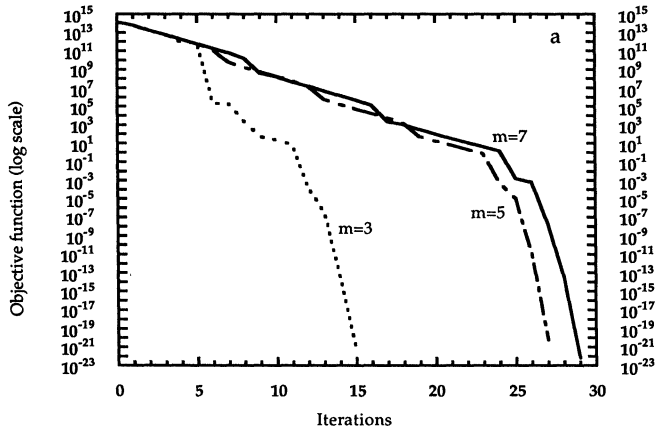
(b)

FIG. 4. *Variation of* (a) *the objective function and* (b) *the norm of gradient with the number of iterations using the* BBVSCG *method with m equal* 7 (*solid*), 5 (*dash dot*), *and* 3 (*dotted*) *for the test library problem* 17.

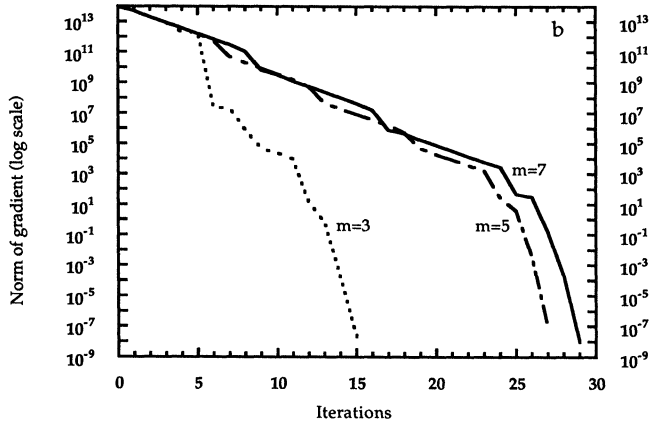
We also find that L-BFGS still requires the fewest function calls among LMQN methods that use nonstandard starting points (data not shown).

Therefore, from the experiments with the 18 library test problems, L-BFGS with a more accurate line search ($\beta = 0.01$) emerges as the most efficient minimizer for problems for which the function calls are inexpensive and the computational effort of the iteration dominates the cost of evaluating the function and gradient. However, both L-BFGS with inexact line searches and CONMIN are very effective on problems for which the function calls are exceedingly expensive. E04DGF does not perform as well as the other LMQN methods.

5.2. Results for the synthetic cluster function problems. For the first one-cluster hyperellipsoidal problem we tested the sensitivity of all the methods to



(a)



(b)

FIG. 5. *Variation of* (a) *the objective function and* (b) *the norm of gradient with the number of iterations using the* BBVSCG *method with m equal* 7 (*solid*), 5 (*dash dot*), *and* 3 (*dotted*) *for the test library problem* 6 *with dimension* 100.

various degrees of ill conditioning by controlling the value of D_1, the dispersion interval in fractional form. Table 5 presents the results for D_1 taken to be 0.2, 0.8, and 0.99, respectively. The corresponding condition numbers are 2.0, 39.9, and 436.8. The results in Table 5 indicate that L-BFGS performs best when the condition number is small. As the condition number is increased, L-BFGS requires the most iterations and function calls, whereas CONMIN-CG uses the fewest function calls. In CPU time E04DGF is most efficient and CONMIN-BFGS is most expensive (even though the latter requires fewer iterations and function calls than does L-BFGS). The full-memory CONMIN-BFGS code spends about four times as much CPU time as does any other method. This occurs because most iteration time is spent in matrix and vector multiplications.

For the second bicluster problem we control the condition number by changing

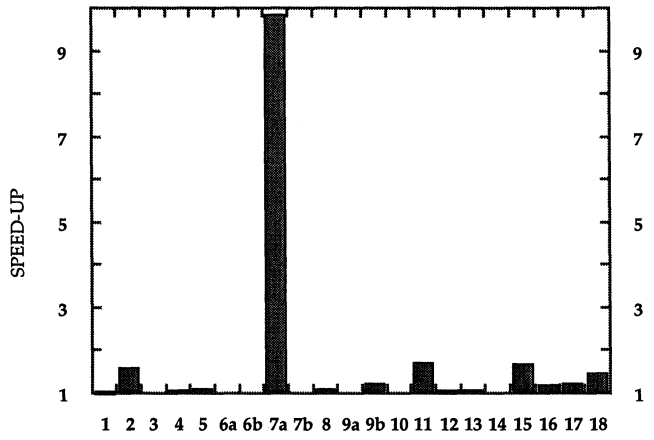


FIG. 6. *Speed-up* NFUN(3)/NFUN(7), *for* L-BFGS *method.*

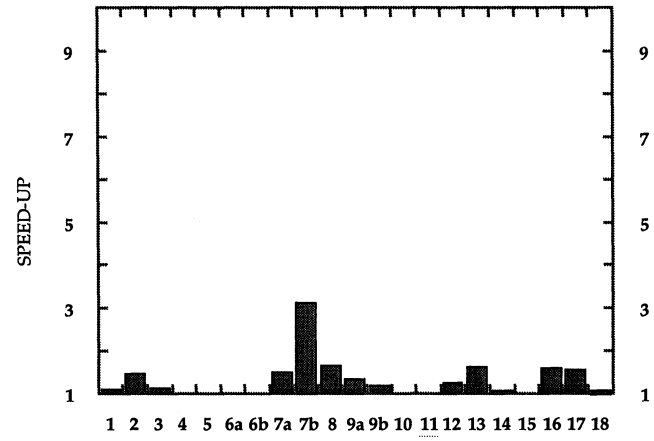


FIG. 7. *Speed-up* NFUN(3)/NFUN(7), *for* BBVSCG *method.*

the position of the center of the second cluster C_2. The performance when the value of the condition number is equal to $8.29, 8.29 \times 10^2$, and 8.29×10^4, respectively, is given in Table 6. We see that when the condition number is equal to 8.29, L-BFGS uses the fewest function calls. However, the differences among the various methods is not significant. When the condition number is increased, L-BFGS again turns out to be the worst. The E04DGF code turns out to be best in all computational respects: number of iterations, number of function calls, and total CPU time. If we use a more accurate line search, L-BFGS is competitive with CONMIN-CG, which is the second best, and is better than BBVSCG.

We also compared the performance of different LMQN methods on a multicluster problem. The same conclusion can be drawn (table omitted): the E04DGF performs best. L-BFGS with a more accurate line search and CONMIN-CG come in second, followed by BBVSCG.

TABLE 3

Eighteen standard library test problems with standard starting points, using the L-BFGS (m = 7) *method with more accurate line search.*

P	N	Iter	Nfun	MTM (total CPU time)	FTM (Function calls' CPU time)
1	3	19	55	0.0325	0.0009
2	6	19	57	0.0352	0.0029
3	3	3	9	0.0038	0.0002
4	2	98	355	0.1938	0.0047
5	3	11	42	0.0199	0.0014
	10	4	14	0.0057	0.0002
6	100	7	27	0.0220	0.0005
	12	36	97	0.1130	0.0479
7	30	254	654	1.2429	0.6812
8	100	58	223	0.2491	0.0039
	10	61	226	0.1371	0.0127
9	50	155	444	0.5163	0.1027
10	2	8	30	0.0130	0.0002
11	4	13	35	0.0219	0.0012
12	3	14	47	0.0348	0.0107
13	100	41	107	0.1656	0.0134
14	100	23	77	0.0880	0.0010
15	100	19	56	0.0685	0.0008
16	2	9	28	0.0139	0.0002
17	4	29	83	0.0507	0.0007
18	100	704	1451	9.8829	7.2431

TABLE 4

Number of wins on the whole set of 18 *test problems with the standard starting points, using limited memory* Q-N *methods.*

WINS	CONMN -CG	E04DGF	L-BFGS (β=0.9)			L-BFGS (β=0.01)	BBVSCG		
			m=3	m=5	m=7	m=7	m=3	m=5	m=7
Iter	6	0	1	0	2	13	1	1	1
Nfun	5	0	2	3	5	2	0	0	5

TABLE 5

One cluster problem (N = 21; K = 1; N_1 = 21; C_1 = 1.0; D_1 = 0.2, 0.8, *and* 0.99), *the condition numbers are* 2.0, 39.9, *and* 436.8, *respectively.*

	m	Iter			Nfun			MTM (total CPU time)		
D_1		0.2	0.8	0.99	0.2	0.8	0.99	0.2	0.8	0.99
CONMIN-CG		10	21	21	21	43	43	0.0157	0.0342	0.0342
Conmin-BFGS		11	45	56	13	47	58	0.0493	0.2187	0.2679
E04DGF		10	21	32	23	45	67	0.0060	0.0119	0.0179
	3	10	50	117	12	56	124	0.0129	0.0654	0.1515
L-BFGS	5	10	45	97	12	52	103	0.0146	0.0716	0.1535
(β=0.9)	7	10	42	99	12	48	107	0.0157	0.0773	0.1855
	3	10	24	44	19	47	87	0.0152	0.0403	0.0747
BBVSCG	5	10	26	26	17	50	49	0.0165	0.0528	0.0527
	7	10	42	61	15	62	101	0.0170	0.0790	0.1269
	3	10	21	45	23	45	46	0.0168	0.0343	0.0347
L-BFGS'	5	10	21	45	23	45	46	0.0185	0.0393	0.0397
($\beta=10^{-2}$)	7	10	21	45	23	45	46	0.0196	0.0436	0.0440

TABLE 6

Bi-cluster problem ($N = 21$; $K = 2$; $N_1 = 11$; $N_2 = 10$; $C_1 = 1.0$; $C_2 = 2.0, 20,$ *and* 200; $D_1 = 0.2, 0.3$), *the condition numbers are* 8.29, 8.29×10^2, *and* 8.29×10^4, *respectively.*

	m	Iter			Nfun			MTM (total CPU time)		
D_1		2.0	20.0	200.	2.0	20.0	200.	2.0	20.0	200.0
CONMIN-CG		18	23	35	37	47	71	0.0318	0.0412	0.0634
Conmin-BFGS		28	90	141	30	92	143	0.1334	0.4516	0.6958
E04DGF		15	19	22	33	42	47	0.0115	0.0148	0.0167
	3	24	158	1009	28	170	1083	0.0338	0.2205	1.4129
L-BFGS	5	24	154	895	29	164	951	0.0392	0.2591	1.5187
(β=0.9)	7	24	168	579	28	182	610	0.0439	0.3330	1.1523
	3	20	67	168	39	129	289	0.0355	0.1198	0.2695
BBVSCG	5	24	75	167	38	128	256	0.0418	0.1406	0.2895
	7	24	81	169	31	126	232	0.0398	0.1636	0.3232
	3	18	21	29	39	53	81	0.0335	0.0419	0.0616
L-BFGS'	5	18	20	29	39	49	83	0.0376	0.0439	0.0699
(β=10^{-2})	7	18	21	30	39	51	88	0.0410	0.0504	0.0824

TABLE 7

Ocean problem ($N = 7330$), *using limited-memory* Q-N *methods.*

Algorithm	Iter	Nfun	MTM (total CPU time)	FTM (function calls' CPU time)
CONMIN-CG	7	15	15.45	15.42
L-BFGS(β=0.9)	22	25	17.15	16.44
BBVSCG	4	8	8.43	9.23
L-BFGS(β=0.001)	22	25	17.03	16.33

5.3. Results for the oceanographic large-scale minimization problem. Only CONMIN-CG, L-BFGS, and BBVSCG were successful for this large-scale problem. E04DGF failed in its line search. All the methods used the same convergence criterion:

$$\|\mathbf{g}_k\| \leq \epsilon, \quad \epsilon = 10^{-8}. \tag{5.1}$$

Numerical experiments indicate that when the number of Q-N updates m is increased from 3 to 7, there is no significant improvement in performance. In Table 7 we present only the results for L-BFGS and BBVSCG when $m = 3$. We see that the function evaluation for this problem is far more expensive than is the iterative procedure. Both L-BFGS and CONMIN-CG require 15 function calls, whereas BBVSCG uses only 8 function calls. Therefore, BBVSCG emerges as the most effective algorithm here. It also uses fewest iterations. No significant improvement was observed for L-BFGS with a more accurate line search.

5.4. Results for the meteorological large-scale minimization problem. Both gradient scaling and nondimensional scaling were applied to the meteorological large-scale minimization problem for all four LMQN methods. CONMIN-CG and BBVSCG failed after the first iteration with either gradient scaling or nondimensional scaling. L-BFGS was successful only with gradient scaling. E04DGF worked only with the nondimensional shallow-water-equations model. It appears that additional scaling is crucial for the success of the LMQN minimization algorithms applied to this real-life, large-scale meteorological problem.

TABLE 8

Meteorological problem with the limited memory quasi-Newton methods.

Control Variables	Algorithm	Iter	Nfun	MTM (total CPUtime)	FTM (function calls' CPU time)
Initial	E04DGF	72	203	36.89	33.56
	L-BFGS	66	89	15.53	14.76
Initial+Boundary	E04DGF	160	481	87.31	79.98
	L-BFGS	179	468	80.70	77.81

TABLE 9

Maximum absolute differences between the retrieval and the unperturbed initial wind and geopotential fields using the limited memory quasi-Newton methods.

Control Variables	Algorithm	$(u^2+v^2)^{1/2}$	ϕ
Initial	E04DGF	0.75E-2	0.12E2
	L-BFGS	0.38E-1	0.90E0
Initial+Boundary	E04DGF	0.10E0	0.64E3
	L-BFGS	0.26E1	0.22E2

Table 8 presents the performance of these two LMQN methods, namely, E04DGF and L-BFGS, when only the initial conditions or the initial-plus-boundary conditions are taken to be the control variables. Because of the different scaling procedures used in the two methods the minimization was stopped when the convergence criterion

$$\|\mathbf{g}_k\| \leq 10^{-4} \times \|\mathbf{g}_0\| \tag{5.2}$$

was satisfied.

We observe from Table 8 that most of the CPU time is spent on function calls rather than in the minimization iteration. By comparing the number of function calls and CPU time we find that the computational cost of L-BFGS is much lower than that of E04DGF. L-BFGS converged in 66 iterations with 89 function calls. In contrast, E04DGF required 72 iterations and 203 function calls to reach the same convergence criterion. This produces rather large differences in the CPU time spent in minimization. L-BFGS uses less than half of the total CPU time required for E04DGF.

The differences between figures showing the retrieved initial wind and geopotential and Fig. 1 are imperceptible (figures omitted). Table 9 gives the maximum differences between the retrieval and the unperturbed initial conditions from E04DGF and L-BFGS minimization results. An accuracy of at least 10^{-3} is reached for both the wind and geopotential fields by using both the codes of L-BFGS and of E04DGF for the initial control. This clearly shows the capability of the unconstrained LMQN methods to adjust a numerical weather prediction model to a set of observations distributed in both time and space.

When we control both the boundary and initial conditions, we expect to produce a much more difficult problem than when we control only the initial conditions. First, since the dimensionality of the Hessian of the objective function is increased by about one order of magnitude (from 10^3 to 10^4), the condition number of the Hessian will increase as $O(N^{2/d})$ [27], where d is the dimensionality of the space variables and N is

TABLE 10

Initial control problem in meteorology.

algorithm	MXITCG	Iter	Nfun	NCG	MTM	FTM
TN1	3	19	20	50	12.20	11.31
	50	20	26	54	13.79	12.89
TN1	3	63	64	170	38.82	37.15
(no prec.)	50	39	40	165	32.78	31.53
TN2	3	81	82	242	68.68	67.21
	50	4	5	91	16.41	16.30

the number of components of the vector of control variables. Second, the perturbation of the boundary conditions creates locally an ill-posed problem. This is reflected by an increase of high-frequency noise near the boundary. In turn, the condition number of the Hessian of the objective function increases.

From Tables 8 and 9 we see, indeed, that when we control both the initial and boundary conditions, minimization becomes much more difficult. The computational cost is doubled and the accuracy of the retrieval is decreased by an order of magnitude compared with those of the initial control problem. The largest differences occur near the boundary for both the wind and geopotential fields. However, the differences between the performances of E04DGF and L-BFGS on the initial- and boundary-value problems are small.

6. Results for TN methods. The meteorology problems of §5.4 were tested for TN1 and TN2. In TN methods performance often depends on the specified maximum number of permitted inner iterations per outer iteration (MXITCG). Our experience suggests that different settings for MXITCG have a small impact on the performance of TN1 but a rather large impact on that of TN2 (see Table 10). This results from our current unpreconditioned implementation for TN2 since the inner CG loop requires more iterations to find a search direction.

To clarify this idea and to see what differences in performance between the two TN methods were due to the different truncation criteria, CG versus Lanczos, and to preconditioning, we also performed minimization for TN1 without diagonal preconditioning. The results are presented in Table 10. Similar trends are identified for both TN1 and TN2 in this case: the cost for large MXITCG is much lower than that for small MXITCG. However, TN2 with MXITCG = 50 performs much better than does TN1 with MXITCG = 50 in terms of Newton iterations, CG iterations, function evaluations, and CPU time. This strongly suggests that with a suitable preconditioner for the problem in meteorology, TN2 might perform best.

Numerical results for both initial control and initial and boundary control are summarized in Tables 11 and 12. We see from the tables that time is approximately proportional to the number of inner iterations. Thus the use of preconditioning in TN1 accelerates performance, as expected. Note that without preconditioning TN1 requires more function evaluations than does TN2. Preconditioning is particularly important as the dimension of the minimization problem increases.

Comparison with Table 9 shows that the TN methods are competitive with L-BFGS. TN1 is better than L-BFGS for initial control and much better than L-BFGS for initial and boundary control. TN1 also produces higher accuracy than do the other three methods (see Tables 9 and 12).

TABLE 11

Meteorological problem with the truncated Newton methods.

Control Variables	Algorithm	Iter	Nfun	MTM (total CPU time)	FTM (function calls' CPU time)
Initial	TN1	19	70	12.20	11.31
	TN2	4	96	16.41	16.30
Initial+Boundary	TN1	70	283	49.96	46.22
	TN2	12	520	87.22	86.30

TABLE 12

Maximum absolute differences between the retrieval and the unperturbed initial wind and geopotential fields using the truncated Newton methods.

Control Variables	Algorithm	$(u^2+v^2)^{1/2}$	ϕ
Initial	TN1	0.89E-2	0.54E2
	TN2	0.58E-2	0.41E2
Initial+Boundary	TN1	0.96E1	0.48E3
	TN2	0.14E0	0.90E3

It appears that for this set of test problems TN methods always require far fewer iterations and fewer function calls than do the LMQN methods. The good performance of the TN methods for large-scale minimization of variational data assimilation problems is very encouraging since minimization is the most computationally intensive part of the assimilation procedure and the numerical weather prediction model already taxes the capability of present-day computers. The complexity of these problem stems from the cost of the integration of the model and of the adjoint system required to update the gradient in the minimization procedure.

7. Summary and conclusions. Four recently available LMQN methods and two TN methods were examined for a variety of test and real-life problems. All methods have practical appeal: they are simple to implement, they can be formulated to require only function and gradient (and possibly additional preconditioning) information, and they can be faster than full-memory Q-N methods for large-scale problems. L-BFGS emerged as the most robust code among the LMQN methods tested. It uses the fewest iterations and function calls for most of the 18 standard test library problems, and it can be greatly improved by a simple scaling or by a more accurate line search. All of the LMQN methods (L-BFGS, CONMIN-CG, and BBVSCG) perform better than the full-memory Q-N BFGS method, especially in terms of total CPU time. E04DGF appears to be the least efficient method for the library test problems. However, numerical results obtained for the synthetic cluster function reveal that E04DGF performs quite well on problems whose Hessian matrices have clustered eigenvalues.

Both variable-storage methods (L-BFGS and BBVSCG) were very successful on the large-scale problem from oceanography, and BBVSCG turned out to perform slightly better on this problem than did L-BFGS.

The convergence rate of the variable-storage methods was accelerated when the number m of Q-N updates was increased for medium-sized problems. However, for small- and large-scale problems both methods showed only a slight improvement as the number of Q-N updates m is increased. The reason for this is not yet known,

and further research is needed. Implementation of these minimization algorithms on vector and parallel computer architectures is expected to yield a significant reduction in the computational cost of large minimization problems.

Only E04DGF and L-BFGS performed successfully on the large-scale optimal control problems in meteorology, and they were successful only after special scalings were applied. L-BFGS performed better than E04DGF in terms of computational cost.

Although the L-BFGS method may be adequate for most present-day large-scale minimization, TN methods yield the best results for large-scale meteorological problems.

Acknowledgments. The authors thank Maria Hopgood for help in testing the 18 problems. The authors also thank Jorge Nocedal and an anonymous reviewer for their insightful remarks, which contributed to a clear and more concise paper.

REFERENCES

[1] A. G. BUCKLEY, *A combined conjugate-gradient quasi-Newton minimization algorithm*, Math. Programming, 15 (1978), pp. 200–210.

[2] ———, *Remark on algorithm* 630, ACM Trans. Math. Software, 15 (1989), pp. 262–274.

[3] A. G. BUCKLEY AND A. LENIR, *QN-like variable storage conjugate gradients*, Math. Programming, 27 (1983), pp. 155–175.

[4] ———, *Algorithm* 630-BBVSCG: *A variable storage algorithm for function minimization*, ACM Trans. Math. Software, 11 (1985), pp. 103–119.

[5] W. C. DAVIDON, *Variable Metric Methods for Minimization*, A.E.C. Research and Development Report ANL-5990, Argonne National Laboratory, Argonne, IL, 1959.

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

[7] J. C. GILBERT AND C. LEMARÉCHAL, *Some numerical experiments with variable-storage quasi-Newton algorithms*, Math. Programming, 45 (1989), pp. 407–435.

[8] 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, CA, 1979.

[9] H. G. GOLUB AND C. VAN LOAN, *Matrix Computation*, 2nd ed., Johns Hopkins Series in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1989.

[10] A. GRAMMELTVEDT, *A survey of finite-difference schemes for the primitive equations for a barotropic fluid*, Mon. Weather Rev., 97 (1969), pp. 387–404.

[11] D. M. LEGLER, I. M. NAVON, AND J. J. O'BRIEN, *Objective analysis of pseudo-stress over the Indian Ocean using a direct minimization approach*, Mon. Weather Rev., 117 (1989), pp. 709–720.

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

[13] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.

[14] NAG, *Fortran Library Reference Manual Mark* 14, No. 3, Numerical Algorithms Group, Downers Grove, IL, 1990.

[15] S. G. NASH, *Newton-type minimization via the Lanczos method*, SIAM J. Numer. Anal., 21 (1984), pp. 770–788.

[16] ———, *Solving nonlinear programming problems using truncated-Newton techniques*, in Numerical Optimization, Proc. SIAM Conference on Numerical Optimization, Society for Industrial and Applied Mathematics, Philadelphia, 1984, pp. 119–136.

[17] ———, *Preconditioning of truncated-Newton methods*, SIAM J. Sci. Statist. Comput, 6 (1985), pp. 599–616.

[18] S. G. NASH AND J. NOCEDAL, *A Numerical Study of the Limited Memory* BFGS *Method and the Truncated-Newton Method for Large Scale Optimization*, Tech. Report NAM 02, Northwestern University, Evanston, IL, 1989.

[19] I. M. NAVON AND D. M. LEGLER, *Conjugate-gradient methods for large-scale minimization in meteorology*, Mon. Weather Rev., 115 (1987), pp. 1479–1502.

[20] I. M. NAVON AND X. ZOU, *Application of the adjoint model in meteorology*, in Automatic Differentiation of Algorithms: Theory, Implementation and Application, A. Griewanek and G. Corliss, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1991, pp. 202–207.

[21] L. NAZARETH, *A relationship between the* BFGS *and conjugate gradient algorithms and its implications for new algorithms*, SIAM J. Numer. Anal., 16 (1979), pp. 794–800; also Tech. Memo ANL-AMD 282, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL, 1976.

[22] ———, *Conjugate gradient methods less dependent on conjugacy*, SIAM Rev., 28 (1986), pp. 501–511.

[23] J. NOCEDAL, *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35 (1980), pp. 773–782.

[24] ———, *Theory of algorithms for unconstrained optimization*, Acta Numerica, 1991, pp. 199–242, p. 340.

[25] A. PERRY, *A Modified Conjugate-Gradient Algorithm*, Discussion Paper 229, Center for Mathematical Studies in Economics and Management Sciences, Northwestern University, Evanston, IL, 1976.

[26] ———, *A Class of Conjugate-Gradient Algorithms with a Two-Step Variable Metric Memory*, Discussion Paper 269, Center for Mathematical Studies in Economics and Management Sciences, Northwestern University, Evanston, IL, 1977.

[27] M. J. D. POWELL, *Restart procedures for the conjugate gradient method*, Math. Programming, 12 (1977), pp. 241–254.

[28] T. SCHLICK, *Modified Cholesky factorizations for sparse preconditioners*, SIAM J. Sci. Comput., 14 (1993), pp. 424–445

[29] T. SCHLICK AND A. FOGELSON, TNPACK—*A truncated Newton minimization package for large-scale problems*, I. *Algorithm and usage*, II. *Implementation examples*, ACM Trans. Math. Software, 18 (1992), pp. 46–111.

[30] T. SCHLICK AND M. L. OVERTON, *A powerful truncated Newton method for potential energy minimization*, J. Comput. Chem., 8 (1987), pp. 1025–1039.

[31] D. F. SHANNO, *On the convergence of a new conjugate gradient algorithm*, SIAM J. Numer. Anal., 15 (1978), pp. 1247–1257.

[32] ———, *Conjugate gradient methods with inexact searches*, Methods Oper. Res., 3 (1978), pp. 244–256.

[33] D. F. SHANNO AND K. H. PHUA, *Remark on algorithm* 500—*a variable method subroutine for unconstrained nonlinear minimization*, ACM Trans. Math. Software, 6 (1980), pp. 618–622.

[34] P. WOLFE, *The secant method for simultaneous nonlinear equations*, Comm. ACM, 2 (1968), pp. 12–13.