SIAM J. SCI. STAT. COMPUT.
Vol. 6, No. 3, July 1985

PRECONDITIONING OF TRUNCATED-NEWTON METHODS*

STEPHEN G. NASH†

Abstract. In this paper we discuss the use of truncated-Newton methods, a flexible class of iterative methods, in the solution of large-scale unconstrained minimization problems. At each major iteration, the Newton equations are approximately solved by an inner iterative algorithm. The performance of the inner algorithm, and in addition the total method, can be greatly improved by the addition of preconditioning and scaling strategies. Preconditionings can be developed using either the outer nonlinear algorithm or using information computed during the inner iteration. Several preconditioning schemes are derived and tested.

Numerical tests show that a carefully chosen truncated-Newton method can perform well in comparison with nonlinear conjugate-gradient-type algorithms. This is significant, since the two classes of methods have comparable storage and operation counts, and they are the only practical methods for solving many large-scale problems. In addition, with the Hessian matrix available, the truncated-Newton algorithm performs like Newton's method, usually considered the best general method for this problem.

Key words. unconstrained optimization, truncated-Newton algorithm, preconditioning strategies, linear conjugate-gradient algorithm

1. Introduction.

The problem of minimizing a real-valued function of n variables

$$\min_x F(x) \tag{1}$$

arises in many contexts and applications. For the purposes of this paper, we assume that F is bounded and twice continuously differentiable. In particular, we are interested in solving problems for which the number of variables is large, but where the gradient of F is available.

The most effective general method for solving (1) is Newton's method, which takes full advantage of first- and second-derivative information about the function F. In its modern, safe-guarded implementations, it provides a standard for measuring the effectiveness of other algorithms. In Newton's method, the direction of search p is computed from the "Newton equations"

$$G^{(k)}p = -g^{(k)}, \tag{2}$$

where $g^{(k)}$ and $G^{(k)}$ are, respectively, the gradient vector and Hessian matrix of second derivatives of F evaluated at the current iterate $x^{(k)}$. When the number of variables n is large, solving (2) can be expensive and can require the storage of an n by n matrix, which may be infeasible. Also, it is necessary to compute $G^{(k)}$ at every iteration. Even for many small problems, this may be very costly. A large-scale problem will often have a sparse Hessian matrix, i.e. the Hessian matrix will have few nonzero entries. This allows Newton's method to be extended to large-scale problems through the use of finite-differencing and sparse-matrix techniques (Powell and Toint (1979)). However, in many contexts (constrained optimization, probability density estimation, etc.) this may not be possible.

Because the Newton equations are based on a Taylor series expansion near the solution x^* of the minimization problem (1), there is no guarantee that the search direction they compute will be as crucial far away from x^*. At the beginning of the solution process, a reasonable approximation to the Newton direction may be almost

* Received by the editors November 30, 1982, and in final revised form March 9, 1984. This research was supported by the National Science Foundation under grants MCS-7926009 and ENG77-06761, and by a postgraduate scholarship from the Natural Sciences and Engineering Research Council of Canada.

† Mathematical Sciences Department, Johns Hopkins University, Baltimore, Maryland 21218.

as effective as the Newton direction itself. It is only gradually, as the solution is approached, that the Newton direction takes on more and more meaning.

These comments suggest that, for large-scale problems, it is sensible to use an iterative method to approximately solve the Newton equations (2). Moreover, it should be an iterative method with a variable tolerance, so that far away from the solution, (2) is not solved to undue accuracy. Only when the solution is approached should we consider expending enough effort to compute something like the exact Newton direction. As we approach the solution, the Hessian $G^{(k)}$ will converge to $G(x^*)$. Consequently, by exploiting information from previous iterations, it is possible that a closer approximation to the exact solution can be determined with no increase in effort.

We will refer to any method which uses an iterative algorithm to approximately solve the Newton equations as a truncated-Newton method. Sherman (1978) suggested using Successive-Over-Relaxation (SOR). This is the simplest of a whole class of methods which have been found to be effective for solving linear systems which arise in partial differential equations. However, it can be difficult to get SOR methods to perform well on general problems. Also, they appear to be prohibitively expensive to use in the context of truncated-Newton methods. The number of linear subiterations required to achieve superlinear convergence increases exponentially at each nonlinear iteration. (Notice that a truncated-Newton method is doubly iterative: there is an outer "nonlinear" iteration to minimize the function $F(x)$, and an inner "linear" iteration to compute a search direction from the Newton equations (2).)

Various authors (Dembo and Steihaug (1983), Garg and Tapia (1980), O'Leary (1982)) have suggested using variants of the linear conjugate-gradient method. Although it is ideal for problems where the coefficient matrix has only a few distinct eigenvalues, it is guaranteed to converge (in exact arithmetic) in at most n iterations for any positive-definite symmetric matrix. Thus, the type of exponential growth mentioned above for SOR-type methods is impossible, at least theoretically.

A requirement of the linear conjugate-gradient method is that the coefficient matrix must be positive definite. Unfortunately, the Hessian matrix is only guaranteed to be positive semidefinite at the solution and may be indefinite elsewhere. Thus, whatever iterative method is chosen to solve (2), it must be able to detect and cope with indefinite systems. This is very closely related to the situation with Newton's method.

The definition of the search direction given by (2) is only satisfactory if $G^{(k)}$ is positive definite. An indefinite $G^{(k)}$ allows the possibility of p not being a descent direction and this may result in convergence to a nonoptimal point. In the context of minimization, it is preferable not to solve the system (2) if $G^{(k)}$ is indefinite. In this case it is better to define p as the solution of a neighbouring positive definite system

$$\bar{G}^{(k)}p = -g^{(k)}, \qquad \bar{G}^{(k)} = G^{(k)} + E.$$

A method for computing $\bar{G}^{(k)}$ when matrix factorizations are feasible is to compute the modified Cholesky factorization of $G^{(k)}$ (Gill and Murray (1974)). The idea of the Gill–Murray algorithm is to increase the diagonal elements of $G^{(k)}$ during the factorization so that the diagonal elements of the factorization are positive and the subdiagonal elements are bounded. An important feature of the Gill–Murray algorithm is the ability to detect that $G^{(k)}$ is not sufficiently positive definite and to compute a satisfactory descent direction nevertheless.

A straightforward application of the linear conjugate-gradient method would not have this property. Moreover, the linear conjugate-gradient algorithm is numerically unstable when applied to an indefinite system. To overcome these difficulties, Nash (1984) has derived a modified linear conjugate-gradient algorithm via the Lanczos algorithm (hereafter referred to as the modified-Lanczos algorithm) which has many of the properties of the Gill–Murray algorithm described above. If the matrix $G^{(k)}$ is sufficiently positive definite, it is identical to the standard linear conjugate-gradient algorithm. A brief description of the modified-Lanczos algorithm appears in § 2.

Our main interest in this paper is the choice of a preconditioning strategy for a truncated-Newton method. In such a method, we solve a sequence of linear systems of the form (2), whose coefficient matrices $G^{(k)}$ will converge to $G(x^*)$ as the solution to the minimization problem is approached. Because of this convergence, it is possible to take advantage of earlier computations to make subsequent linear systems easier to solve, and hence to improve the efficiency of the overall algorithm. In large problems where it is expensive to compute information, it is especially important to make as much use as possible of every computed quantity. This is generally accomplished by using current information to precondition future iterations.

Preconditioning is such a powerful and general idea that there exist preconditioned versions of almost every known numerical algorithm, both direct and iterative. Direct algorithms often use preconditioning to reduce the error in the computed solution. One common example of this is the use of column scaling in Gaussian elimination (see, for example, Wilkinson (1965, Chap. IV)). Iterative methods generally use preconditioning to accelerate convergence, although they may also be concerned with the condition of the problem. One of the best known and best understood examples of this is the generalized (i.e. preconditioned) linear conjugate-gradient algorithm of Concus, Golub and O'Leary (1976).

To conclude this section, we give here a description of a truncated-Newton method in algorithmic form. The details of the methods used to iteratively solve the Newton equations and to precondition the algorithm will be given later.

TRUNCATED-NEWTON ALGORITHM.

TN0. Given $x^{(0)}$, some initial approximation to x^*. Set $k=0$.

TN1. If $x^{(k)}$ is a sufficiently accurate approximation to the minimizer of F, terminate the algorithm.

TN2. Approximately solve the Newton equations (2) using some iterative algorithm with preconditioning $M^{(k)} \simeq G^{(k)}$. (See §§ 2 and 3.)

TN3. With the search direction p computed in step TN2, find $\alpha > 0$ such that $F(x^{(k)} + \alpha p) < F(x^{(k)})$. (Line search; see below.)

TN4. Set $x^{(k+1)} = x^{(k)} + \alpha p$, $k = k+1$. Go to step TN1.

For the purposes of this paper, we will assume that a modified-Lanczos algorithm (see below) will be used in Step TN2 to approximately solve the Newton equations. In step TN3, the line search, $F(x)$ must be "sufficiently decreased" in order to guarantee convergence to a local minimum. One approach is to ensure that the search direction p is a descent direction ($p^T g^{(k)} < 0$), that $|g(x^{(k)} + \alpha p)^T p| \leqq -\eta p^T g^{(k)}$ with $0 \leqq \eta < 1$, and that $F(x^{(k)}) - F(x^{(k)} + \alpha p) \geqq -\mu p^T g^{(k)}$ where $0 < \mu \leqq \frac{1}{2}$, $\mu < \eta$. By choosing μ small (say 10^{-4}), almost any α that satisfies the first condition will also satisfy the second. The step-length α can be computed using safeguarded polynomial interpolation (Gill and Murray (1979)).

2. The modified-Lanczos method.

The general form of the modified-Lanczos algorithm is outlined below. If the Hessian matrix G is positive definite, this method is equivalent to the linear conjugate-gradient algorithm of Hestenes and Stiefel (1952). For more complete details, refer to Nash (1984).

Assume temporarily that G is positive definite. Recall that we are iteratively solving

$$Gp = -g. \tag{3}$$

We use the Lanczos algorithm (Lanczos (1950)) to compute a tridiagonal matrix that is an orthogonal projection of G. At stage q of the algorithm:

$$V_q^T G V_q = T_q, \qquad V_q^T V_q = I, \tag{4}$$

where V_q is an n by q orthogonal matrix, and T_q is a q by q tridiagonal matrix. The tridiagonal matrix T_q is factored into its Cholesky factors:

$$T_q = L_q D_q L_q^T,$$

where D_q is diagonal with positive diagonal entries, and L_q is lower bidiagonal with ones on the main diagonal (this factorization is only possible if T_q is positive definite). This factorization is then used to compute p_q (the qth approximation to the solution of (3)):

$$T_q y_q = L_q D_q L_q^T y_q = (-V_q^T g), \qquad p_q = V_q y_q.$$

(Paige and Saunders (1975) have derived iterative formulas for p_q based on this derivation.) At each inner iteration q, the direction p_q is tested to see if it "adequately" solves the Newton equations (3); if so, the inner iteration is *truncated*, and the search direction p is defined as p_q (see § 4 for details). The sequence of iterates p_q is the same as that generated by the linear conjugate-gradient algorithm, if we choose $V_1 = (g/\|g\|_2)$.

The linear conjugate-gradient algorithm can only be safely used when G is positive definite, whereas the Lanczos algorithm only requires that G be symmetric. The above derivation will enable us to adapt the linear conjugate-gradient algorithm to indefinite systems, as we now indicate.

If g contains a component of the negative eigenspace of G, then indefiniteness in G will ultimately show up in T_q (for $q = n$, (4) defines an orthogonal similarity transformation). In fact, due to the properties of the Lanczos algorithm, it will show up fairly early (Parlett (1980)). O'Leary (1982) has suggested applying the modified-Cholesky factorization of Gill and Murray (1974) to T_q in this case. However, this factorization requires information about the complete matrix T_n in order to ensure stability, information not available to this iterative algorithm.

Because T_q is iteratively generated, and T_{q-1} is a principal submatrix of T_q, it is possible to determine exactly the stage q at which T_q becomes indefinite. If this occurs, Nash (1984) suggests boosting the diagonal elements of the lower 2×2 diagonal block so that the resulting matrix $\bar{T}_q$ is positive definite. Because only this 2×2 is modified, the iterative nature and the low storage requirements are unchanged. In addition, the size of any diagonal modification is bounded by $3(\delta + \gamma_q + \zeta_q)$ where γ_q and ζ_q are the largest (in absolute value) diagonal and off-diagonal elements of T_q, and δ is a tolerance for zero (used to bound T_q away from singularity).

2.1. Properties of the search direction. Even if the Hessian is indefinite, the approximate solutions p_q of the Newton equations will be descent directions for the minimization algorithm, i.e. $p_q^T g < 0$ for $q > 0$. If $\bar{T}_q$ denotes the (possibly modified) tridiagonal matrix computed above, then

$$g^T p_q = -g^T V_q \bar{T}_q^{-1} V_q^T g < 0$$

if $V_q^T g \neq 0$, since $\bar{T}_q$ is positive definite by construction. Since v_1, the first column of V_q, will be chosen as a nonzero multiple of $M^{-1}g$ for some positive definite matrix M, $V_q^T g$ will be nonzero, and hence p_q will be a descent direction as desired.

Although necessary to guarantee the convergence of the algorithm, the fact that p is a descent direction is not enough to ensure that it is an effective search direction. It should also be well-scaled, i.e. a unit step along p should approximate the minimum of the function in that direction. Near x^*, this will be true for Newton's method, but cannot be guaranteed for nonlinear conjugate-gradient methods. However, regardless of how many modified-Lanczos iterations are used to compute the search direction p, a truncated-Newton method will generally give a well-scaled search direction, in the sense described below.

If the line-search procedure described in § 1 is used, the primary test (for an approximate minimum along the direction p) is $|g(x+\alpha p)^T p| \leqq -\eta g^T p$, where $0 \leqq \eta < 1$. Assuming that $V_q^T G V_q$ is positive definite, setting $\alpha = 1$ (in the hope of a well-scaled direction), and using a Taylor series expansion, we obtain

$$\begin{aligned} p_q^T g(x+p_q) &= -g^T V_q (V_q^T G V_q)^{-1} V_q^T g \\ &\quad + g^T V_q (V_q^T G V_q)^{-1} (V_q^T G V_q)(V_q^T G V_q)^{-1} V_q^T g + O(\|g\|^3) \\ &= O(\|g\|^3). \end{aligned}$$

This final expression, representing the cubic remainder term in the Taylor series, will be small when $x^{(k)}$ is near to x^*, or when $F(x)$ is approximated well by a quadratic function. In these cases, we can expect that the search direction from the truncated-Newton method will be well-scaled, even after only one inner iteration. (See also Dembo and Steihaug (1983).)

2.2. Preconditioning. If a matrix M is available such that $M \simeq G$, then the modified-Lanczos algorithm can take advantage of this information. The algorithm is applied (implicitly) to the equivalent system of linear equations

$$(M^{-1/2} G M^{-1/2}) M^{1/2} p = -M^{-1/2} g.$$

The number of iterations required to solve this transformed system is equal to the number of distinct eigenvalues of $M^{-1}G$. In addition, ignoring this finite-termination property, the algorithm converges linearly with rate $(\kappa^{1/2}-1)/(\kappa^{1/2}+1)$, where κ is the condition number of $M^{-1}G$ in the 2-norm. We aim to choose M so that $\kappa(M^{-1}G)$ is small, and so that $M^{-1}G$ has fewer distinct eigenvalues than G, thus making the system of equations easier to solve. In practice, the matrix $M^{-1/2}$ is not formed; all that is required is that a system of equations of the form

$$My = c$$

be solved at each step. For details of these results, see Concus, Golub and O'Leary (1976).

2.3. Matrix/vector products. At each iteration, the Lanczos algorithm requires the computation of a matrix/vector product Gv involving the Hessian matrix G. However, the matrix G is not required explicitly. If G is explicitly available, these matrix/vector products can be formed directly. If G is sparse, a finite-difference approximation to G could be formed and used to compute them (see Thapa (1980)). Otherwise, Gv could be approximated by finite-differencing along the gradient g:

$$G(x)v \simeq \frac{g(x+hv)-g(x)}{h}$$

for some suitably chosen small value of h (see, for example, Gill and Murray (1974), O'Leary (1982)).

3. Preconditioning strategies. With truncated-Newton methods, there are two principal ways in which a preconditioning strategy can be selected. A basic preconditioning might be chosen using the formulas for some low-memory nonlinear algorithm; this is the subject of § 3.1. Secondly, this nonlinear algorithm might be further preconditioned by some scaling of the variables. This is the subject of § 3.2. In either case, our goal is to develop a preconditioning operator dynamically, as the problem is being solved, and not to rely on a priori information.

3.1. Preconditioning based on a nonlinear algorithm. A truncated-Newton algorithm operates by using some iterative algorithm to approximately solve a sequence of equations of the form

$$G^{(k)}p=-g^{(k)}.$$

As the solution is approached, it might be hoped that information gained solving equation (k) might assist in solving equation $(k+1)$. This information is generally used by forming, either explicitly or implicitly, a matrix $M\simeq G^{(k+1)}$. The better M approximates $G^{(k+1)}$, the better the preconditioning strategy will be (see § 2.2). In order to use M within the modified-Lanczos algorithm, M must be positive definite, and linear systems involving M should be "easy" to solve. For example, the matrix M might be diagonal or in factored form.

It is possible to design preconditioning strategies by exploiting ideas from other minimization methods. Most nonlinear optimization algorithms can be viewed as computing a search direction by solving, possibly implicitly, a system of linear equations

$$Bp=-g$$

with some operator B, where B is an approximation to the Hessian G. By applying the formulas for the nonlinear method to any vector (instead of $-g$), we implicitly define a preconditioning matrix M.

The optimal choice would be $M=G^{(k+1)}$ (Newton's method) since the inner iteration would then converge instantly; however, the costs in storage and computation would be prohibitive. Setting $M=I$, i.e. using an unpreconditioned algorithm, corresponds to preconditioning with the steepest-descent operator; this is simple to use, but not particularly effective. As a compromise, Nash (1984) has suggested using the operator from a limited-memory quasi-Newton method, which is inexpensive to use, and yet still effective at improving the performance of the inner algorithm.

The class of limited-money quasi-Newton methods (see Gill and Murray (1979)) define the search direction as a linear combination of the gradient vector and a subset of the previous search directions. They generalize nonlinear conjugate-gradient algorithms, and are suitable for problems in which the Hessian cannot be stored.

These methods derive their name from the class of quasi-Newton methods for unconstrained optimization. The direction of search for a quasi-Newton method can be defined as

$$p=-H_kg^{(k)},$$

where H_k is an $n\times n$ matrix which is stored explicitly and is an approximation to the inverse Hessian matrix. After computing the change in x, $s_k\equiv x^{(k+1)}-x^{(k)}$ and the corresponding change in the gradient vector, $y_k\equiv g^{(k+1)}-g^{(k)}$, the approximate Hessian is updated to include the new curvature information obtained during the k-th iteration. For example, the BFGS formula for H_{k+1} is given by

$$H_{k+1}=H_k+\frac{(s_k-H_ky_k)s_k^T+s_k(s_k-H_ky_k)^T}{y_k^Ts_k}-\frac{(s_k-H_ky_k)^Ty_k}{(y_k^Ts_k)^2}s_ks_k^T \tag{5}$$

(see Dennis and Moré (1977)). If exact linear searches are made and F is a positive-definite quadratic function, the matrix H_{k+1} satisfies the so-called quasi-Newton condition for k pairs of vectors $\{s_j, y_j\}$, i.e.,

$$s_j = H_{k+1} y_j, \qquad j = 1, 2, \cdots, k.$$

In this case, if the Hessian of F is G, then $G s_j = y_j$ and consequently

$$s_j = H G s_j,$$

and the matrix HG has k unit eigenvalues with eigenvectors $\{s_j\}$.

Limited-memory quasi-Newton methods define the direction of search as $-Hg^{(k)}$; the matrix H is never stored explicitly; rather, only the vectors $\{s_j, y_j\}$ that define the rank-one corrections are retained (see Shanno (1978), Gill and Murray (1979), and Nocedal (1980)).

Different methods can be developed by varying the number of vectors $\{s_j, y_j\}$ stored and the choice of quasi-Newton updating formula. For example, if we define the matrix H to be the identity matrix updated by one iteration of the BFGS formula (5), and if exact line searches are performed, the algorithm will be equivalent to the Fletcher–Reeves nonlinear conjugate-gradient method.

When no preconditioning is used, the first linear iterate will be a multiple of the steepest-descent direction, which is often a poor approximation to the Newton direction. Preconditioning with an effective nonlinear algorithm offers the hope that the first iterate will approximate the Newton direction quite well, and that an adequate search direction can be computed using only a few inner iterations.

3.2. Diagonal scaling of the variables. Nonlinear minimization algorithms have been found to work more efficiently if the variables are properly scaled. In part, this means that a unit step along the search direction will approximate the minimizer of the function in that direction. It also implies that the tolerances for the algorithm have the correct scaling; this is a factor even for the more scale-invariant algorithms such as Newton's method. One way of achieving this is through a diagonal scaling matrix. In this context, the inverse of this diagonal matrix will be used as the initial approximation to the matrix H of § 3.1.

There is some theoretical evidence to indicate that, among diagonal scalings, the most effective strategy will be to approximate the diagonal of the Hessian. Forsythe and Straus (1955) have shown that if the Hessian matrix G is two-cyclic, then the diagonal of G is the optimal diagonal preconditioning. This assumption is valid for many problems arising in partial differential equations. Also, in the general case, van der Sluis (1969) has proven that preconditioning with the diagonal of G will be nearly optimal, in the sense that the condition number (in the 2-norm) of G preconditioned by its diagonal will be at most n times as large as the condition number of the optimally diagonally preconditioned G. Thus, estimating the diagonal of G should be effective for all problems.

The sample scaling strategies derived here will be based on quasi-Newton approximations to the diagonal of the Hessian matrix. It is possible to use the direct form of the BFGS formula (5) to approximate the diagonal of G. Here we approximate G by a sequence of matrices B_q, rather than approximating G^{-1} by matrices H_q.

Because the linear conjugate-gradient algorithm is equivalent to the BFGS algorithm (when applied to the same quadratic objective function with $B_0 = I$), it is possible to show that $B_n = G$, if G is positive definite and if the iteration does not terminate prematurely (see Nazareth (1979)). Thus, if we were able to update only the diagonals of B, at the end of n steps we would have the exact values for the diagonal elements of G.

To develop this diagonal update, we will ignore the nonlinear algorithm for the moment, and concentrate our attention on one instance of the linear conjugate-gradient method. We are attempting to minimize the quadratic function

$$\phi(p)=\tfrac{1}{2}p^TGp+p^Tc$$

and hence

$$g(p)=\nabla\phi(p)=Gp+c=-r(p),$$

where $r(p)$ is the residual at p. The linear conjugate-gradient algorithm is initialized with $p_0=0$, and at the qth iteration, the next estimate of the solution is computed as

$$p_{q+1}=p_q+\alpha_q u_q,$$

where u_q is the search direction and α_q is the step-length.

The BFGS algorithm computes the (same) search direction using the formula

$$B_q u_q=-g_q, \tag{6}$$

where $g_q\equiv g(p_q)$. If an exact line-search is used, the step-length for the BFGS algorithm is that same as that for the linear conjugate-gradient algorithm. Under the assumptions that $p_0=0$, $B_0=I$, and that the new approximate Hessian B_{q+1} is computed using the direct form of the BFGS formula (5)

$$B_{q+1}=B_q-\frac{1}{s_q^TB_qs_q}(B_qs_q)(B_qs_q)^T+\frac{1}{y_q^Ts_q}y_qy_q^T, \tag{7}$$

both algorithms compute the same estimates of the solution at every stage.

It is possible to adapt (7) so that only the diagonal of the update need be computed. Using (6) and

$$s_q=p_{q+1}-p_q=\alpha_q u_q,$$

we can conclude that

$$B_qs_q=-\alpha_q g_q. \tag{8}$$

The other important fact is

$$y_q=g_{q+1}-g_q=\alpha_q Gu_q. \tag{9}$$

If we incorporate (8) and (9) in (7), we obtain

$$B_{q+1}=B_q-\frac{1}{u_q^Tr_q}r_qr_q^T+\frac{1}{u_q^T(Gu_q)}(Gu_q)(Gu_q)^T. \tag{10}$$

(These quantities are all computed within the conjugate-gradient algorithm.) Using (10), any individual element of B_q can be individually updated. However, when used to compute a scaling matrix, only the diagonal of B_q will be formed.

When the linear conjugate-gradient algorithm is used in its standard form, (10) is quite adequate. However, using instead the preconditioned modified-Lanczos algorithm (see § 2) creates two further problems. First, in practice, a new scaling matrix will be generated using an iteration preconditioned by some operator M. In this case, the BFGS algorithm should be initialized with $B_0=M$. To see this, replace G by $M^{-1/2}GM^{-1/2}$ in the above derivation.

A second problem arises because the linear conjugate-gradient algorithm is implicitly implemented using the modified-Lanczos algorithm: only constant multiples of the search direction u_q and the residual r_q are computed. These multiplicative factors do not affect the final term in (10), since the factors enter equally into the numerator and the denominator. The other rank-one matrix is affected. However, the true residual can be computed using $r_q = \alpha_q \tilde{v}_q$, where $\tilde{v}_q$ is the unnormalized current Lanczos vector (Parlett (1980)). This leaves only the inner product $u_q^T r_q$. Using the recurrence relation for the search direction u_q, and the fact that the residuals are M-orthogonal, it can be shown that

$$u_q^T r_q = r_q^T M^{-1} r_q = \alpha_q^2 \tilde{v}_q^T M^{-1} \tilde{v}_q.$$

Note that $M^{-1}\tilde{v}_q$ is computed within the modified-Lanczos algorithm.

Because the Hessian matrix is not always positive definite, the modified-Lanczos algorithm alters the subproblem it is solving when it runs across evidence of indefiniteness. The preconditioning scheme is trying to approximate the diagonal of the actual Hessian matrix, and the preconditioning algorithm described above has the property of hereditary positive definiteness, so there is some question as to what should be done when the Hessian matrix is modified. We have chosen to omit the diagonal update whenever the matrix goes indefinite, in order to ensure that B_q remains positive definite.

Using (10) it is possible to compute any number of subdiagonals in addition to the main diagonal. Because this extension is so straightforward, the details will be omitted here.

An additional possibility is to use exact information about the diagonal of the Hessian either to precondition the linear algorithm or to initialize the linear preconditioning. Note, however, that even if matrix-vector products of the form Gv can be found, it may be inconvenient to compute G_{ii}. Also, away from the solution of the minimization problem, the matrix G may be indefinite, so that the diagonal of the Hessian may not define a positive definite preconditioning matrix. In that case, some rule for modifying negative diagonal elements would have to be derived.

4. Numerical results.

In this section we compare the numerical behavior of threee truncated-Newton algorithms with that of other methods. The methods tested are:

1. Algorithm PLMA—A two-step BFGS limited-memory quasi-Newton method with a simple diagonal scaling. PLMA is the most successful nonlinear conjugate-gradient-type method tested in the survey of Gill and Murray (1979).
2. Algorithm MNA—A modified Newton method using first and second derivatives (Gill and Murray (1974)).
3. Algorithm QNM—A quasi-Newton method using the full n by n BFGS update of the approximate Hessian matrix (Gill and Murray (1972)).
4. Algorithm TN—A truncated-Newton algorithm, implemented via the modified-Lanczos algorithm, and preconditioned with PLMA with the simple diagonal scaling replaced by the diagonal of (10).
5. Algorithm BTN—A (basic) truncated-Newton algorithm, implemented via the linear conjugate-gradient algorithm, and with no preconditioning strategy.
6. Algorithm PBTN—Algorithm BTN, but preconditioned using the diagonal of (10).

Eighteen problems are considered. Of these, 11 problems are of dimension 50 or less, and 7 problems are of dimension 100. The test examples may be separated into two classes. The first class contains problems whose Hessian matrix at the solution has clustered eigenvalues; the second contains problems whose Hessian matrix has an arbitrary eigenvalue distribution.

Example 1. Pen 1 (Gill, Murray and Pitfield (1972)).

$$F(x) = a \sum_{i=1}^{n} (x_i - 1)^2 + b\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{4} \right)^2.$$

The solution varies with n, but $x_i = x_{i+1}$, $i = 1, \cdots, n-1$. All the runs were made with $a = 1$, $b = 10^{-3}$. With these values, the Hessian matrix at the solution has $n-1$ eigenvalues $O(1)$ and one eigenvalue $O(10^{-3})$. The Hessian matrix is full and consequently, for large values of n, conjugate-gradient type methods are the only techniques available.

Example 2. Pen 2 (Gill, Murray and Pitfield (1972)).

$$F(x) = a \sum_{i=2}^{n} \left((e^{x_i/10} + e^{x_{i-1}/10} - c_i)^2 + (e^{x_i/10} - e^{-1/10})^2 \right)$$
$$+ b\left(\left(\sum_{i=1}^{n} (n - i + 1) x_i^2 - 1 \right)^2 + \left(x_1 - \frac{1}{5} \right)^2 \right)^2,$$

where $c_i = e^{i/10} + e^{(i-1)/10}$ for $i = 2, \cdots, n$. The solution varies with n, but $x_i = x_{i+1}$ for $i = 1, \cdots, n-1$. This example was also run with $a = 1$ and $b = 10^{-3}$. For these values the Hessian matrix at the solution has $n-2$ eigenvalues $O(1)$ and two eigenvalues $O(10^{-3})$. The Hessian matrix is full.

Example 3. Pen 3 (Gill, Murray and Pitfield (1972)).

$$F(x) = a\left\{ 1 + e^{x_n} \sum_{i=1}^{n-2} (x_i + 2x_{i+1} + 10x_{i+2} - 1)^2 \right.$$
$$+ \left(\sum_{i=1}^{n-2} (x_i + 2x_{i+1} + 10x_{i+2} - 1)^2 \right)\left(\sum_{i=1}^{n-2} (2x_i + x_{i+1} - 3)^2 \right)$$
$$\left. + e^{x_{n-1}} \sum_{i=1}^{n-2} (2x_i + x_{i+1} - 3)^2 \right\}$$
$$+ \left(\sum_{i=1}^{n} (x_i^2 - n) \right)^2 + \sum_{i=1}^{n/2} (x_i - 1)^2.$$

At the minimum, this function has $n/2$ eigenvalues $O(1)$ and $n/2$ eigenvalues $O(10^{-3})$. The Hessian matrix is full.

The remaining examples have arbitrary distributions of eigenvalues at the solution.

Example 4. Chebyquad (Fletcher (1965)).

$$F(x) = \sum_{i=1}^{n} f_i(x)^2,$$

where

$$f_i(x) = \int_0^1 T_i^*(x)\, dx - \frac{1}{n} \sum_{j=1}^{n} T_i^*(x_j), \qquad i = 1, \cdots, n,$$

and $T_i^*(x)$ is the ith-order shifted Chebyshev polynomial. The Hessian matrix is full.

Example 5. GenRose. This function is a generalization of the well-known two-dimensional Rosenbrock function (Rosenbrock (1960)). For $n > 2$,

$$F(x) = 1 + \sum_{i=2}^{n} \left(100(x_i - x_{i-1}^2)^2 + (1 + x_i)^2 \right).$$

Our implementation of this function differs from most others in that $F(x)$ is unity at the solution rather than zero. This modification ensures that the function cannot be computed with unusually high accuracy at the solution and is therefore more typical of practical problems.

The next three examples arise from the discretization of problems in the calculus of variations. Similar problems arise in the numerical solution of optimal control problems. The general continuous problem is to find the minimum of the functional

$$J(x(t)) = \int_0^1 f(t, x(t), x'(t))\,dt,$$

over the set of piecewise differentiable curves with the boundary conditions $x(0) = a$, $x(1) = b$. If $x(t)$ is expressed as a linear sum of functions that span the space of piecewise cubic polynomials then minimization of J becomes a finite-dimensional problem with a block tridiagonal Hessian matrix. The piecewise polynomials are assumed to be in C^1, and equally spaced knots are used.

Example 6. Cal 1 (Gill and Murray (1973)).

$$J(x(t)) = \int_0^1 \{x(t)^2 + x'(t)\tan^{-1} x'(t) - \log(1 + x'(t)^2)^{1/2}\}\,dt,$$

with the boundary conditions $x(0) = 1$, $x(1) = 2$.

Example 7. Cal 2 (Gill and Murray (1973)).

$$J(x(t)) = \int_0^1 \{100(x(t) - x'(t)^2)^2 + (1 - x'(t))^2\}\,dt,$$

with the boundary conditions $x(0) = x(1) = 0$.

Example 8. Cal 3 (Gill and Murray (1973)).

$$J(x(t)) = \int_0^1 \{e^{-2x(t)^2}(x'(t)^2 - 1)\}\,dt,$$

with the boundary conditions $x(0) = 1$, $x(1) = 0$.

Example 9. QOR (Toint (1978)).

$$F(x) = \sum_{i=1}^{50} \alpha_i x_i^2 + \sum_{i=1}^{33} \beta_i \left(d_i - \sum_{j\in A(i)} x_j + \sum_{j\in B(i)} x_j \right)^2,$$

where the constants α_i, β_i, d_i and sets $A(i)$ and $B(i)$ are described in Toint's paper. This function is convex with a sparse Hessian matrix.

Example 10. GOR (Toint (1978)).

$$F(x) = \sum_{i=1}^{50} c_i(x_i) + \sum_{i=1}^{33} b_i(y_i),$$

where

$$c_i(x_i) = \begin{cases} \alpha_i x_i \log_e(1 + x_i), & x_i \geqq 0, \\ -\alpha_i x_i \log_e(1 + x_i), & x_i < 0, \end{cases}$$

$$y_i = d_i - \sum_{j\in A(i)} x_j + \sum_{j\in B(i)} x_j$$

and

$$b_i(y_i) = \begin{cases} \beta_i y_i^2 \log_e(1 - y_i), & y_i \geqq 0, \\ \beta_i y_i^2, & y_i < 0. \end{cases}$$

The constants α_i, β_i, d_i and sets $A(i)$ and $B(i)$ are defined as in Example 9. This function is convex but there are discontinuities in the second derivatives.

Example 11. ChnRose (Toint (1978)).

$$F(x)=1+\sum_{i=2}^{25}(4\alpha_i(x_{i-1}-x_i^2)^2+(1-x_i)^2),$$

where the constants α_i are those used in Example 9. The value of $F(x)$ at the solution has been modified as in Example 5. The Hessian matrix is tridiagonal.

The starting points used were the following:

Start 1 $x^{(0)}=(0,0,\cdots,0)^T$.

Start 2 $x^{(0)}=\left(\frac{1}{n+1},\frac{2}{n+1},\cdots,\frac{n}{n+1}\right)^T$.

Start 3 $x^{(0)}=(1,-1,1,-1,\cdots)^T$.

Start 4 $x^{(0)}=(-1,-1,\cdots,-1)^T$.

4.1. Details of the algorithms. All the routines are coded in double precision FORTRAN IV. The run were made on an IBM 370/168, for which the relative machine precision ε is approximately 10^{-15}.

The truncated-Newton routines require the computation of matrix/vector products of the form $G^{(k)}v$. For routine TN with Examples 5-11, sparse finite-differencing techniques (Thapa (1980)) were used to approximate $G^{(k)}$ at the beginning of each major iteration, and this approximation was used to compute the matrix/vector products. The difference parameter used here was $\varepsilon^{1/2}$, where ε is the machine precision. Elsewhere, the matrix/vector products were computed by differencing the gradient along the vector v (§ 2.3). Because our interest is in methods that do not require second derivatives, tests were not made using exact second-derivative information.

For all truncated-Newton algorithms, a fairly stringent criterion was used to terminate the modified-Lanczos iterations. Following Dembo and Steihaug (1983), the modified Lanczos iterations are terminated after $n/2$ iterations, or if

$$\frac{\|r_q\|}{\|g^{(k)}\|}\leqq \min\{1/k, \|g^{(k)}\|\},$$

where r_q is the qth residual of the linear system. This criterion forces the algorithm to behave like a conjugate-gradient algorithm near the beginning of the iteration and like Newton's method near the solution. We stress, however, that when second derivatives are not available, or the cost of the matrix/vector product $G^{(k)}v$ is high, a criterion must be used that always leads to a small number of linear iterations. Because the computation of the search direction can be degraded by loss of orthogonality, at most $n/2$ modified Lanczos iterations were allowed at each major step.

Each problem was solved using three values of η, the step-length accuracy (see § 1); these values were 0.25, 0.1, and 0.001. Each algorithm requires two additional user-specified parameters. The first (λ) limits the change in x at each iteration (the quantity $\|x^{(k+1)}-x^{(k)}\|_2$). The value of λ was set at 10 for all problems to avoid overflow during the computation of the objective function. The second parameter is an estimate of the value of the objective function at the solution and is used to compute the initial step for the step-length algorithm. In each case, this parameter was set to the value of $F(x)$ at the solution.

The results are contained in Tables 1-3. Each table entry refers to the iteration at which

$$F^{(k)}-F(x^*)<10^{-5}(1+|F(x^*)|),$$

where x^* is the solution.

TABLE 1
(Reprinted from Nash [14].)

Function	η	PLMA		MNA		QNM		BTN		PBTN		TN		Finite-difference	Lanczos iterations	Totals
Cal 1	.25	194	366	7	9	162	191	16	978	8	227	10	18	60	181	78/199
Start 1	.1	204	401	6	11	89	214	17	1151	8	232	9	19	54	139	73/158
$n = 50$	.001	205	456	6	17	88	269	13	787	7	252	9	34	54	132	88/166
Cal 2	.25	64	106	4	4	28	52	8	133	7	104	7	8	42	61	50/69
Start 1	.1	61	118	4	4	28	54	8	133	7	104	7	8	42	61	50/69
$n = 50$	.001	60	123	4	6	28	67	8	135	7	107	7	10	42	61	52/71
Cal 3	.25	80	152	6	6	90	114	7	206	7	125	7	8	42	104	50/112
Start 1	.1	78	155	5	7	59	135	7	206	7	127	7	8	42	104	50/112
$n = 50$	.001	77	161	5	11	59	161	8	289	7	125	7	17	42	96	59/113
Cal 1	.25	423	819	not	run	not	run	26	3855	9	828	10	19	60	390	79/409
Start 1	.1	429	854	not	run	not	run	23	3214	8	570	10	25	60	484	85/409
$n = 100$	.001	416	905	not	run	not	run	26	3948	8	602	10	35	60	337	95/372
Cal 2	.25	112	204	not	run	not	run	8	256	7	172	8	9	48	108	57/117
Start 1	.1	107	206	not	run	not	run	8	256	7	172	8	9	48	108	57/117
$n = 100$	.001	113	228	not	run	not	run	8	259	8	199	8	11	48	111	59/122
Cal 3	.25	143	270	not	run	not	run	9	508	7	196	7	8	42	151	50/159
Start 1	.1	142	281	not	run	not	run	9	636	7	196	7	9	42	159	51/168
$n = 100$	.001	138	284	not	run	not	run	9	617	7	195	7	18	42	158	60/176
GenRose	.25	108	201	62	202	128	287	33	499	42	584	31	75	93	255	168/330
Start 2	.1	119	263	66	257	118	323	32	518	36	469	33	99	99	249	198/348
$n = 50$	.001	119	330	88	392	118	412	35	614	37	468	34	143	102	252	245/395
GenRose	.25	191	365	not	run	not	run	60	1150	73	1055	57	164	171	420	335/684
Start 2	.1	192	410	not	run	not	run	62	1153	72	1012	60	190	180	585	370/775
$n = 100$	.001	188	528	not	run	not	run	60	1197	63	1062	58	248	174	534	422/782
Chebyquad	.25	38	75	29	121	32	65	10	104	10	87	7	16	37	37	53/53
Start 2	.1	33	71	24	116	28	67	9	96	9	105	8	17	51	51	68/68
$n = 20$	.001	33	90	30	161	28	92	11	164	10	129	9	29	61	61	90/90

For paired entries (n_1, n_2), n_1 is the number of major iterations required, and n_2 is the number of function/gradient evaluations used.

TABLE 2

Function	η	PLMA		MNA		QNM		BTN		PBTN		TN		Finite-difference	Lanczos iterations	Totals
Pen 1	.25	22	53	17	18	27	33	10	46	10	46	7	18	11	11	29/29
Start 3	.1	8	27	9	25	8	26	10	48	10	48	7	19	11	11	30/30
$n = 50$	.001	8	32	7	29	8	31	6	40	6	40	3	15	4	4	19/19
Pen 2	.25	52	118	17	17	134	242	14	89	10	60	11	35	29	29	64/64
Start 3	.1	28	76	9	31	99	322	13	86	10	64	12	42	36	36	78/78
$n = 50$	.001	15	71	6	26	73	341	12	118	9	67	12	57	38	38	95/95
Pen 3	.25	40	76	40	44	67	135	10	62	11	60	10	14	33	33	47/47
Start 3	.1	38	76	12	44	63	150	10	61	9	52	9	16	30	30	46/46
$n = 50$	.001	28	71	11	48	56	155	10	72	10	63	9	24	30	30	54/54
Pen 1	.25	17	40	not	run	not	run	2	11	2	11	2	9	2	2	11/11
Start 3	.1	2	9	not	run	not	run	2	11	2	11	2	9	2	2	11/11
$n = 100$	.001	2	10	not	run	not	run	2	12	2	12	2	10	2	2	12/12
Pen 2	.25	14	28	not	run	not	run	6	25	6	26	6	15	14	14	29/29
Start 3	.1	7	18	not	run	not	run	5	24	5	23	6	18	12	12	30/30
$n = 100$	.001	7	29	not	run	not	run	5	36	5	33	6	26	15	15	41/41
Pen 3	.25	49	85	not	run	not	run	13	79	11	75	11	24	41	41	65/65
Start 3	.1	48	94	not	run	not	run	13	89	11	75	11	26	41	41	67/67
$n = 100$	.001	35	83	not	run	not	run	11	91	11	89	11	34	43	43	77/77
QOR	.25	14	29	3	3	23	39	5	29	5	25	6	7	48	18	55/25
Start 1	.1	14	29	3	3	13	27	5	29	5	25	6	7	48	18	55/25
$n = 50$	.001	14	29	3	3	13	27	5	29	5	25	6	7	48	18	55/25
GOR	.25	14	71	5	5	29	59	7	74	7	77	7	9	56	51	65/60
Start 1	.1	14	76	5	5	29	59	7	74	7	77	7	10	56	51	66/61
$n = 50$	.001	42	97	5	7	29	72	6	61	7	85	7	17	56	50	73/67
ChnRose	.25	40	82	15	28	48	97	10	88	11	121	11	23	33	54	56/77
Start 4	.1	37	76	16	48	46	122	10	94	11	91	11	25	33	50	58/75
$n = 25$	.001	43	119	12	47	47	164	11	104	1	4	14	56	42	70	98/126

For paired entries (n_1, n_2), n_1 is the number of major iterations required, and n_2 is the number of function/gradient evaluations used.

TABLE 3

Total	PLMA	MNA	QNM	BTN	PBTN	TN	Finite-difference	Lanczos iterations	Totals
Smaller functions $n \leq 50$	1,944 4,276	541 1,755	1,895 4,604	383 7,217	353 4,275	347 910	1,505	2,446	2,415/3,356
Larger functions $n = 100$	2,775 5,750	not run	not run	367 17,427	331 6,614	307 916	1,147	3,717	2,063/4,633
Grand totals	4,719 10,026	--	--	750 24,644	684 10,889	654 1,826	2,652	6,163	4,478/7,989

For paired entries (n_1, n_2), n_1 is the number of major iterations required, and n_2 is the number of function/gradient evaluations used.

4.2. Discussion of results. With the exception of the results for TN, each entry is a pair of numbers: the first is the number of major iterations; the second is the number of function/gradient evaluations required to solve the problem (for BTN and PBTN, this reflects both the line search and the matrix/vector products). For TN, more detailed results are given. The first pair of numbers gives the total number of iterations, and the number of function/gradient evaluations used in the line search. The finite-difference column records the number of gradient evaluations used to compute the matrix/vector products. The next column is the total number of modified Lanczos iterations (each iteration will normally be dominated by the cost of the matrix/vector product, comparable to a gradient evaluation). The final column combines the line-search cost with the inner-iteration cost to give a measure of the total cost of the minimization; two totals are given: the first combines the line-search costs with finite-differencing costs, and the second with the inner-iteration costs.

We first compare the truncated-Newton algorithms among themselves. It is clear that Algorithm TN is superior to the other two. This is not surprising due to the more elaborate preconditioning strategies that it uses. Based on the total number of iterations required, TN is only marginally better than the other two routines. But based on the number of function/gradient evaluations, there is a clear difference. Even without taking advantage of sparsity, PBTN is 36% slower than TN, and BTN is over three times as slow. A comparison of PBTN with BTN indicates the improvement even simple preconditioning strategies can make to a truncated-Newton algorithm. The two routines are identical, except that PBTN has a diagonal scaling as a preconditioner. This addition is inexpensive (three extra vectors are needed), but offers a better than 50% improvement in performance (based on the total number of function/gradient evaluations).

To compare truncated-Newton algorithms with other methods, we will use the results for Algorithm TN. In the following, results for the Newton algorithm MNA and the quasi-Newton method QNM are only available for the smaller functions ($n \leqq 50$). Tests with the larger functions ($n = 100$) were not made due to the storage and computational costs. The total number of function/gradient evaluations (Table 3) will be the primary factor for comparison.

A comparison of TN with the limited-memory quasi-Newton algorithm PLMA shows that TN is 50% better if sparsity of the Hessian is taken into account, and 20% better otherwise (i.e., if each Lanczos iteration requires a gradient evaluation to approximate the matrix/vector product). This comparison is important, since these two classes of methods have comparable storage and operation counts, and they are the only practical methods for solving many large-scale problems.

A comparison of TN with the quasi-Newton method QNM on the smaller test functions indicates that TN is 50% better if sparsity is exploited, and 30% better otherwise. QNM, unlike TN, requires matrix manipulation, and hence has higher storage and operation counts than TN. Both algorithms only require first derivative information.

A comparison of TN with the modified Newton method MNA, again on the smaller test functions, shows that MNA is 30%-50% better than TN, depending on whether sparsity is exploited. However, MNA computes, stores and factors the Hessian matrix, and this is not reflected in the scores for MNA. For this reason, a further comparison is suggested, using the "TN" rather than the "totals" column in Table 3. The "TN" column does not reflect the cost of the matrix/vector products, i.e., the "second-derivative costs" of the truncated-Newton algorithm. From this point of view, TN is twice as efficient as MNA. This is surprising, since the truncated-Newton method is a compromise on Newton's method designed to enable the solution of large-scale problems.

The results of comparisons for individual functions are not always so remarkable. For many of the functions in Table 2, the Hessian has clustered eigenvalues. All of these problems tend to be easy to solve, and there are few striking differences in performance. The remaining problems (Table 1) have more arbitrary eigenvalue distributions, and are considerably harder to solve. Here, the simple truncated-Newton Algorithm BTN has particular difficulties (Cal 1, $n = 50$, 100). Even Newton's method (MNA) appears to struggle with some functions (GenRose, $n = 50$); and performs worse that any other routine in one case (Chebyquad, $n = 20$). For these two functions, the Hessian is frequently indefinite, suggesting that complete modified factorizations are not always an effective treatment for nonconvex functions.

Acknowledgments. The author would like to thank his thesis advisors Philip Gill, Gene Golub, and Walter Murray for their many helpful suggestions. Also, thanks to Mukund Thapa for kindly providing the subroutines for computing the sparse Hessian matrices of the test examples.

BIBLIOGRAPHY

[1] P. CONCUS, G. GOLUB AND D. P. O'LEARY, *A generalized conjugate-gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. Bunch and D. Rose, eds., Academic Press, New York, 1976, pp. 309-332.

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

[3] J. E. DENNIS AND J. J. MORÉ, *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19 (1977), pp. 46-89.

[4] R. FLETCHER, *Function minimization without evaluating derivatives—a review*, Comput. J., 8 (1965), pp. 33-41.

[5] G. E. FORSYTHE AND E. G. STRAUS, *On best conditioned matrices*, Proc. Amer. Math. Soc., 6 (1965), pp. 340-345.

[6] N. K. GARG AND R. A. TAPIA, QDN: *A variable storage algorithm for unconstrained optimization*, Department of Mathematical Sciences Report, Rice Univ., Houston, 1980.

[7] P. E. GILL AND W. MURRAY, *Quasi-Newton methods for unconstrained optimization*, J. Inst. Maths. Applics., 9 (1972), pp. 91-108.

[8] ———, *The numerical solution of a problem in the calculus of variations*, in Recent Mathematical Developments in Control, D. J. Bell, ed., Academic Press, New York, 1973, pp. 97-122.

[9] ———, *Newton-type methods of unconstrained and linearly constrained optimization*, Math. Prog., 17 (1974), pp. 311-350.

[10] ———, *Conjugate-gradient methods for large-scale nonlinear optimization*, Report SOL 79-15, Operations Research Dept., Stanford Univ., Stanford, CA, 1979.

[11] P. E. GILL, W. MURRAY AND R. A. PITFIELD, *The implementation of two revised quasi-Newton algorithms for unconstrained optimization*, Report NAC 11, National Physical Laboratory, England, 1972.

[12] M. HESTENES AND E. STIEFEL, *Methods of conjugate-gradients for solving linear systems*, J. Res. NBS, 49 (1952), pp. 409-436.

[13] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. NBS, 45 (1950), pp. 255-282.

[14] S. G. NASH, *Newton-type minimization via the Lanczos algorithm*, SIAM J. Numer. Anal., 21 (1984), pp. 770-778.

[15] 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.

[16] D. P. O'LEARY, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Prog., 23 (1983), pp. 20-33.

[17] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617-629.

[18] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[19] M. J. D. POWELL AND P. L. TOINT, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060-1074.

[20] H. H. ROSENBROCK, *An automatic method for finding the greatest or least value of a function*, Comput. J., 3 (1960), pp. 175-184.

[21] D. F. SHANNO, *Conjugate gradient methods with inexact searches*, Math. Oper. Res., 3 (1978), pp. 244-256.

[22] A. H. SHERMAN, *On Newton-iterative methods for the solution of systems of nonlinear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 755-771.

[23] M. THAPA, *Optimization of unconstrained functions with sparse Hessian matrices*, Ph.D. thesis, Dept. Operations Research, Stanford Univ., Stanford, CA, 1980.

[24] P. L. TOINT, *Some numerical results using a sparse matrix updating formula in unconstrained optimization*, Math. Comp., 32 (1978), pp. 839-851.

[25] A. VAN DER SLUIS, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1979), pp. 14-23.

[26] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London, 1965.