AD-A078 713 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 12/1
CONJUGATE-GRADIENT METHODS FOR LARGE-SCALE NONLINEAR OPTIMIZATI--ETC(U)
OCT 79 P E GILL , W MURRAY DAAG29-79-C-0110
UNCLASSIFIED SOL-79-15 ARO-16470.1-M NL

1 OF 1
AD-
A078713

END
DATE
FILMED
1-80
DDC

ARO 16470.1-M

(2)

Systems
Optimization
Laboratory

LEVEL II

Department of Operations Research
Stanford University
Stanford, CA 94305

(18) ARO

(19) 16470.1-M

(2)

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
Stanford University
Stanford, California
94305

(9) Technical rept.,

(6)

CONJUGATE-GRADIENT METHODS FOR LARGE-SCALE NONLINEAR OPTIMIZATION.

by

(10) Philip E. Gill and Walter Murray

(14) TECHNICAL REPORT SOL-79-15
(11) October 1979
(12) 64

Research and reproduction of this report were partially supported by the Department of Energy Contract DE-AS03-76-SF00326, PA No. DE-AT-03-76ER72018; the National Science Foundation Grants MCS76-20019 A01 and ENG77-06761; and the U.S. Army Research Office Contract DAAG29-79-C-0110.

(15)

408765

/p

Conjugate-Gradient Methods For Large-scale Nonlinear Optimization†

Philip E. Gill and Walter Murray
Systems Optimization Laboratory
Department of Operations Research
Stanford University
Stanford, CA 94305

ABSTRACT

In this paper we discuss several recent conjugate-gradient type methods for solving large-scale nonlinear optimization problems. We demonstrate how the performance of these methods can be significantly improved by careful implementation. A method based upon iterative preconditioning will be suggested which performs reasonably efficiently on a wide variety of significant test problems.

Our results indicate that nonlinear conjugate-gradient methods behave in a similar way to conjugate-gradient methods for the solution of systems of linear equations. These methods work best on problems whose Hessian matrices have sets of clustered eigenvalues. On more general problems, however, even the best method may require a prohibitively large number of iterations. We present numerical evidence that indicates that the use of theoretical analysis to predict the performance of algorithms on general problems is not straightforward.

†Invited paper, the Institute of Management Sciences XXIV International Meeting, Hawaii, June 1979.

1. Introduction

This report describes the progress of a search for an efficient algorithm to find the minimum of a general nonlinear function subject to upper and lower bounds upon the variables. In particular we are interested in solving problems for which the number of variables n is very large, say of the order of several hundred.

Our interest in the solution of bound-constrained problems as opposed to unconstrained problems stems from two observations. Firstly, it is rare for there to be no constraints at all upon the individual variables. Secondly, even if the solution does not lie upon any of the bounds, their presence can prevent the function from being evaluated at unreasonable or nonsensical points. Intuitively, one would expect that it would be possible to construct a more efficient algorithm by providing more information about the region in which the solution is expected to lie. For some classes of algorithm this is indeed the case, but we shall show in later sections that the presence of bounds upon the variables may *adversely* affect the performance of conjugate-gradient methods.

The most successful algorithms for bound-constrained minimization proceed as follows. At any stage of the algorithm the variables are partitioned into two sets: the set of *fixed* variables which are at their upper or lower bounds, and the set of *free* variables which are currently being optimized. An unconstrained minimization is performed with respect to the free variables. This unconstrained problem is altered occasionally if a free variable violates a bound or a fixed variable is allowed to become free. (For the precise details of how the free variables are selected, see Gill and Murray, 1976). Clearly, bound-constrained minimization is closely related to unconstrained minimization and this is reflected in the content of this paper.

Probably the most commonly-used techniques for minimizing a general unconstrained nonlinear function are the class of *quasi-Newton methods* (see Dennis and Moré, 1977, for a survey). However, such methods require the storage of an $n \times n$ approximate Hessian matrix of second derivatives and as n becomes large these methods become impractical. The first algorithm that could be applied specifically to large-scale unconstrained optimization was due to Fletcher and Reeves (1964), their algorithm being a generalization of the Hestenes and Stiefel conjugate-gradient method for solving the linear equations $Ax = b$ for a positive-definite symmetric $n \times n$ matrix A. The Hestenes and Stiefel algorithm is iterative and if no rounding error is made, requires n iterations or fewer to find a solution. A fundamental advantage of the method is that no matrix storage is required over and above the storage of the problem itself.

The work of Hestenes and Stiefel was motivated in part by a misconception that prevailed in the early 1950's concerning the numerical stability of direct

methods for solving linear equations. It was believed that the rounding error involved in solving even small systems of equations would always prevent an accurate answer from being found. It was expected that an iterative procedure such as the conjugate-gradient method would prove to be inherently more stable because an inaccurate iterate would automatically be refined in subsequent iterations. Ironically the opposite is true; the conjugate-gradient method is far more susceptible to rounding error than a typical direct method. Moreover, rounding error may cause the algorithm to require many more than n iterations to find the solution. This feature of the algorithm is particularly disappointing, especially when it can be argued that even n is an excessive number of iterations for large problems. However, even when finite-precision arithmetic is being used, there are linear equations which can be solved in far fewer than n iterations. These problems tend to have coefficient matrices whose eigenvalues are clustered into sets containing eigenvalues of similar magnitude.

The development of conjugate-gradient methods for optimization closely paralleled that of conjugate-gradient methods for linear equations in the sense that initial enthusiasm for the method was quickly dispelled by disappointing numerical performance. Extensive testing during the late 1960's and early 1970's showed that the Fletcher-Reeves algorithm was generally inferior to the best of the alternative methods whenever the storage of an $n \times n$ matrix was not an impediment to the application of alternative methods. During the last few years there have been significant improvements in the design of conjugate-gradient algorithms (we shall discuss some of these improvements in this paper), but in many cases these have been more than outweighed by improvements in the rival techniques. In particular there has been considerable interest in modified-Newton and quasi-Newton methods for nonlinear problems with sparse Hessian matrices (Gill and Murray, 1973; Curtis, Powell and Reid, 1974; Toint, 1977).

However, there is a class of unconstrained problems for which conjugate-gradient methods are currently the *only* techniques that can be applied. This is the class of problems for which the Hessian matrix is very large, but not sparse. Such problems arise in large-scale linearly-constrained and nonlinearly-constrained minimization (see Gill and Murray, 1974b; Murray and Wright, 1978). In this situation the Hessian matrices are of the form Z^TGZ where G and Z are large matrices which may be sparse, but whose product Z^TGZ is large and dense.

In Section 2 we discuss the Fletcher-Reeves conjugate-gradient method and its recent improvements by Beale and others. We demonstrate how the performance of these algorithms can be significantly improved by careful implementation. In Section 3 we consider the class of limited-memory quasi-Newton methods suggested by Perry (1977) and Shanno (1978a). This is followed by a discussion of methods which attempt at each iteration to precondition the problem so that the rate of

convergence of conjugate-gradient and limited-memory quasi-Newton methods can be improved. A diagonal preconditioning technique is suggested which can be used improve the performance of most conjugate-gradient type methods.

Finally, in Section 7 we describe some extensive numerical tests which indicate that nonlinear conjugate-gradient methods behave very similarly to conjugate-gradient methods for linear equations. Problems whose Hessian matrices at the solution contain sets of clustered eigenvalues may be minimized in significantly fewer than n iterations. Problems without this property may require anything from between n and $5n$ iterations, with approximately $2n$ iterations or fewer being a common figure for moderately difficult problems. The numerical results suggest that a preconditioning based upon a diagonal scaling may lead to significant improvements in performance on general problems.

2. The traditional conjugate-gradient method and its modifications

2.1 The Fletcher-Reeves algorithm

The Fletcher-Reeves conjugate-gradient algorithm for minimizing a general nonlinear function $F(x)$ proceeds as follows. Let x_0 be a given starting point and let k denote the current iteration, starting with $k = 0$. The iteration requires g_k, the gradient vector $\nabla F(x)$ evaluated at x_k, the k-th estimate of the minimum. At each iteration a vector p_k (known as the *direction of search*) is computed and the new estimate x_{k+1} is given by $x_k + \alpha_k p_k$ where α_k (the *step length*) minimizes the function $F(x_k + \alpha_k p_k)$ with respect to the scalar α_k. During the first iteration, p_k is just the steepest-descent direction $-g(x_0)$. On completion of the k-th linear minimization, the direction of search for the next iteration is found from the formula

$$p_{k+1} = -g_{k+1} + \beta_k p_k, \tag{2.1}$$

where

$$\beta_k = \frac{\|g_{k+1}\|_2^2}{\|g_k\|_2^2}, \tag{2.2}$$

and $\|u\|_2$ denotes the Euclidean norm of a vector u.

When we are minimizing a quadratic function $F(x) = c^T x + \frac{1}{2} x^T Q x$ with Q a symmetric positive-definite matrix and c an n-vector, the directions obtained from the Fletcher-Reeves algorithm are identical to those of the Hestenes and Stiefel conjugate-gradient method for solving the linear equations $Qx = -c$ (Hestenes and Stiefel, 1952). In the quadratic case the step length α_k can be computed in closed form as $\alpha_k = -g_k^T p_k / p_k^T Q p_k$; moreover it can be shown (see, for example,

Fletcher, 1972) that the directions of search are mutually conjugate, i.e.

$$p_i^T Q p_j = 0, \qquad i \neq j,$$

and that the set of gradient vectors $\{g_i\}$ are mutually orthogonal, i.e.

$$g_i^T g_j = 0, \qquad i \neq j.$$

It is relatively easy to demonstrate that the conjugacy of the set of directions gives n-step termination on quadratic functions. If we make the transformation $x = Pu$, where P is the matrix with columns equal to the directions $p_0, p_1, \ldots, p_{n-1}$, then the transformed quadratic function $\phi(u) = F(Pu)$ is separable in the variables u and can be minimized by successively minimizing ϕ with respect to each of the variables u_j in turn.

The formula for β_k used by Fletcher and Reeves was one of several suggested by Hestenes and Stiefel. Theoretically these formulae are equivalent for a quadratic function and Hestenes and Stiefel sought to choose a formula with the best properties when the computation is subject to rounding error. The first step in the derivation of all the formulae for β_k is the recognition that the concept of *conjugacy* may be replaced by one of *orthogonality*. For a quadratic function the vector y_k, which is defined as the difference between the gradients at any two iterates x_{k+1} and x_k, is given by

$$y_k = g_{k+1} - g_k = Q(x_{k+1} - x_k) = \alpha_k Q p_k.$$

Consequently the conjugacy condition $p_i^T Q p_j = 0$ is equivalent to the orthogonality condition $y_i^T p_j = 0$. The most obvious formula for β_k follows from pre-multiplying (2.1) by y_k and choosing β_k such that $y_k^T p_{k+1} = 0$, i.e.

$$\beta_k = y_k^T g_{k+1} / y_k^T p_k. \tag{2.3}$$

If we make use of the fact that $g_{k+1}^T p_k$ and $g_k^T p_{k-1}$ vanish if an exact linear search is made then

$$\begin{aligned} y_k^T p_k &= -g_k^T p_k \\ &= -g_k^T(-g_k + \beta_{k-1} p_{k-1}) \\ &= \|g_k\|_2^2. \end{aligned}$$

This leads to the formula given by Polak and Ribière (see Polak, 1971):

$$\beta_k = y_k^T g_{k+1} / \|g_k\|_2^2. \tag{2.4}$$

Finally, since the gradients are mutually orthogonal for a quadratic function we have $y_k^T g_{k+1}$ equal to $\|g_{k+1}\|_2^2$, giving (2.2).

For general nonlinear functions the alternative formulae for β_k are no longer equivalent. We prefer to use (2.3) since y_k is orthogonal to p_{k+1} irrespective of the accuracy of the linear search or any possible non-quadratic behavior of the objective function. Moreover, Powell (1977) has shown that the use of (2.2) may cause slow convergence in the general nonlinear case when exact linear searches are made. The orthogonality of y_k to p_{k+1} in the absence of an exact linear search is particularly important because it allows the traditional conjugate-gradient algorithm to be generalized to perform inexact linear searches. The ability to use inexact linear searches is a prerequisite for any algorithm designed to minimize bound-constrained problems.

The finite termination property of conjugate-gradient methods on quadratic functions motivated Fletcher and Reeves to abandon the use of (2.1) after a cycle of n linear searches and set p_{k+1} as the steepest-descent direction, $-g_{k+1}$. This strategy is known as *restarting* or *resetting*. Restarting with the steepest-descent direction is based upon the questionable assumption that the reduction in $F(x)$ along the restart direction will be greater than that obtained if the usual formula were used.

We shall refer to the conjugate-gradient algorithm that uses (2.3) for the definition of β_k and restarts with the steepest-descent direction every n iterations as the *traditional conjugate-gradient method.*

2.2 The traditional conjugate-gradient method with inexact linear searches

The difficulty and cost of finding the exact minimum of $F(x)$ along p_k have resulted in many implementations of the traditional conjugate-gradient method allowing values of α_k which do not necessarily give a zero directional derivative $g(x_k + \alpha_k p_k)^T p_k$. For example, in the implementation of Fletcher and Reeves α_k is computed by taking increasing multiples of a scalar until a point is reached with a positive directional derivative. This point and the latest point at which the directional derivative is negative are then used as a basis for cubic interpolation. This interpolation is continued until a point is obtained at which the function value is less than $F(x_k)$.

For a quasi-Newton method there are essentially two conditions that must be satisfied if convergence is to be guaranteed. The first condition is that the function $F(x)$ must be sufficiently reduced at each iteration. The second condition is that the search direction must not become arbitrarily close to being orthogonal to the steepest-descent direction; this is equivalent to requiring that $-g_k^T p_k / \|g_k\|_2 \|p_k\|_2$

is greater than a constant that is bounded away from zero.

The first condition is satisfied for any α_k computed by the following step-length algorithm.

Step-length algorithm QNSL

Let $\{\alpha^j\}$ define a sequence of points that tend in the limit to the minimum of $F(x)$ along p_k. (If $F(x)$ is smooth this sequence can be computed by means of some safeguarded polynomial interpolation algorithm.) Let t be the index of the first member of this sequence such that

$$|g(x_k + \alpha^t p_k)^T p_k| \leq -\eta g_k^T p_k, \tag{c1}$$

where η $(0 \leq \eta < 1)$ is some constant scalar. Let μ $(0 < \mu \leq \frac{1}{2})$ be another constant scalar. Find the smallest non-negative integer r such that

$$F_k - F(x_k + 2^{-r}\alpha^t p_k) \geq -2^{-r}\alpha^t \mu g_k^T p_k \tag{c2}$$

and set $\alpha_k = 2^{-r}\alpha^t$. ∎

If α_k is computed according to this rule it can be shown (Gill and Murray, 1973) that

$$F_k - F(x_k + \alpha_k p_k) > \phi\left(\frac{-g_k^T p_k}{\|p_k\|}\right), \tag{2.5}$$

where ϕ is a function such that, for any sequence $\{c_k\}$,

$$\lim_{k\to\infty} \phi(c_k) = 0 \text{ implies } \lim_{k\to\infty} c_k = 0. \tag{2.6}$$

An important property of conditions (c1) and (c2) is that if μ is chosen as a small value (say 10^{-4}) then, unless $F(x)$ is a pathologically ill-behaved function, any value α^t satisfying (c1) automatically satisfies (c2) with $r = 0$. In this case the step-length algorithm reduces to finding a scalar α_k such that

$$|g(x_k + \alpha_k p_k)^T p_k| \leq -\eta g_k^T p_k.$$

The value of η can be specified by the user and can be used to give a step length that is well-suited to the problem being solved. If η is chosen as 0.9 the algorithm will generally compute a "crude" value of α_k, provided it satisfies (c2). This value is usually α^0, the value used to start the iterative method that computes the minimum of $F(x)$ along p_k. If η is chosen as zero, α_k will be the point that minimizes $F(x)$ along p_k.

It is important to note that a step-length algorithm based upon the inequalities

$$|g(x_k + \alpha_k p_k)^T p_k| \leq -\eta g_k^T p_k \quad \text{and} \quad F_k - F_{k+1} \geq -\mu \alpha_k g_k^T p_k$$

for fixed values of μ and η is not satisfactory since no member of the sequence $\{\alpha^j\}$ may satisfy these inequalities simultaneously.

It is a feature of both Newton-type and quasi-Newton methods that, as the iterates approach the solution, the unit step will invariably satisfy both the conditions (c1) and (c2). This property is useful because $\alpha^0 = 1$ can be used to initiate the sequence $\{\alpha^j\}$. If by chance unity is not an acceptable value for α_k, the next interpolated value generally will be. This property of quasi-Newton methods implies that it usually requires an average of between one and two function evaluations only per iteration to obtain overall convergence.

If we are to apply a similar step-length algorithm to conjugate-gradient methods, it is necessary to add an additional condition on α^t. If (2.1), the formula for p_{k+1}, is pre-multiplied by g_{k+1}^T we find that

$$g_{k+1}^T p_{k+1} = -g_{k+1}^T g_{k+1} + \beta_k g_{k+1}^T p_k. \tag{2.7}$$

If an exact linear search is made, $g_{k+1}^T p_k$ is zero and the direction p_{k+1} must be a descent direction with respect to g_{k+1} since $g_{k+1}^T p_{k+1}$ is just the negative quantity $-g_{k+1}^T g_{k+1}$. However, if an arbitrary inexact linear search is made, the point x_{k+1} may be such that $\beta_k g_{k+1}^T p_k$ is positive and larger than $-g_{k+1}^T g_{k+1}$. Consequently, there is the possibility that p_{k+1} will not be a descent direction since $g_{k+1}^T p_{k+1}$ may be positive or zero. A typical remedy for such an eventuality is to restart the algorithm with p_{k+1} as the steepest-descent direction. Consequently, a "crude" step-length may cause the efficiency of the algorithm to be severely impaired by the use of a large number of steepest-descent iterations.

This problem is easily overcome if algorithm QNSL is used with an additional condition on the termination of the sequence $\{\alpha^j\}$. The following algorithm is based on a useful property of conjugate-gradient methods whereby the initial directional derivative for the next iteration can be computed relatively cheaply during the computation of the sequence $\{\alpha^j\}$.

Step-length algorithm TCGSL

Let σ be a small positive scalar and $\bar{g}_{k+1}$, $\bar{p}_{k+1}$ and $\bar{\beta}_k$ denote the quantities g_{k+1}, p_{k+1} and β_k respectively, computed at the point $x_k + \alpha^t p_k$. Compute α^t as in algorithm QNSL but with the additional condition that

$$-\bar{g}_{k+1}^T \bar{p}_{k+1} \geq \sigma \|\bar{g}_{k+1}\|_2 \|\bar{p}_{k+1}\|_2. \tag{c3}$$

If this condition is not satisfied we proceed with computing the sequence $\{\alpha^j\}$ until it is satisfied or the minimum along p_k is found. ∎

Note that $\bar{p}_{k+1}$ need not be computed explicitly, since $\bar{g}_{k+1}^T\bar{p}_{k+1}$ can be computed by using $\bar{g}_{k+1}^T\bar{g}_{k+1}$, $\bar{\beta}_k$ and $\bar{g}_{k+1}^Tp_k$. If the first value of α^t satisfies the condition (c3) then there is no extra cost involved in this test since these quantities are used elsewhere in the algorithm.

A desirable feature for any step-length algorithm is the facility for imposing an upper bound λ upon the step length; that is, the final α_k must satisfy $\alpha_k\|p_k\|_2 \leq \lambda$. This bound may be used in several ways: to prevent overflow in the user-supplied function routines (see Section 7, for example); to increase efficiency by ensuring that $F(x)$ is evaluated only at sensible values of x; to prevent the step-length algorithm from returning an inordinately large step because no smaller step would satisfy the convergence criterion; to guarantee that the new point x_{k+1} remains feasible when the step-length routine is being used for constrained minimization. It is clear that the convergence of an algorithm that relies upon an exact linear search in the computation of α_k will be impeded if the distance to the bound is less than the step to the minimum. For the traditional conjugate-gradient method, the implications are more serious because the computation of the restricted step may prevent p_{k+1} from being a descent direction. From (2.7) it is clear that if $\beta_k g_{k+1}^Tp_k$ is positive then $g_{k+1}^Tp_{k+1}$ may also be positive. If α_k is restricted to be less than some value λ_k (say), there may be no acceptable value of α for which $g_{k+1}^Tp_{k+1}$ is negative.

Another disadvantage of the traditional conjugate-gradient method with inexact linear searches is that the algorithm will be efficient only for problems for which the gradient vector can be computed with approximately the same cost or less as the objective function. If $g(x)$ is expensive to compute, then the sequence $\{\alpha_k\}$ should be computed by a method utilizing function values only; for example, if $F(x)$ is smooth, the sequence may be computed by means of safeguarded quadratic interpolation. Under these circumstances condition (c1) should be replaced by

$$F(x_k+\nu p_k) - F(x_k+\alpha^t p_k) \leq -\eta(\nu-\alpha^t)g_k^Tp_k,$$

where ν is any estimate of the optimal α such that $\nu < \alpha^t$ (see Gill and Murray, 1974c). Unfortunately, we cannot avoid the computation of the gradient at the point $x_k+\alpha^tp_k$, as it is required to check that condition (c3) holds. This implies that a relatively accurate linear search must always be made within a conjugate-gradient method that has been adapted to accept finite-difference approximations to the derivatives. This will impair the efficiency of the algorithm on those problems for which the cost of a function evaluation dominates the cost of an iteration.

It is a property of conjugate-gradient methods that a_k often varies enormously in magnitude, even when the relative change in the objective function is very small. This implies that a sensible choice of a^0 is crucial if the algorithm is to be efficient. In general, the direction of search rarely approximates the Newton direction and consequently the unit step is a poor choice for a^0. (It is precisely because of this that the user should be encouraged to place reasonable bounds upon the variables whenever possible. The use of bounds in this way is likely to prove even more effective for conjugate-gradient methods than it is for quasi-Newton methods.)

A choice of initial step length that we have found most successful is one suggested by Davidon (1959):

$$a^0 = \begin{cases} -2(F_k - F_{est})/g_k^T p_k, & \text{if } -2(F_k - F_{est})/g_k^T p_k \leq 1; \\ 1, & \text{if } -2(F_k - F_{est})/g_k^T p_k > 1, \end{cases} \tag{2.8}$$

where F_{est} is a user-specified estimate of the function value at the solution. (It may appear unreasonable to expect the user to provide such an estimate, but in our experience, it is rare for a user to have absolutely no idea of the value of the function at the solution. If F_{est} is not specified, the software has the facility for always choosing the unit step length for a^0.) In many situations the use of (2.8) was essential in order to avoid overflow during the computation of the objective function during the first iteration. A unit step along the steepest-descent direction will often compute the function at very large values of x if the function is badly scaled.

Alternatives tried were

$$a^0 = -\frac{2(F_{k-1} - F_k)}{g_k^T p_k},$$

and a value of a^0 which was increased or decreased from that of the previous iteration according to the value of a_{k-1}. However, since neither of these estimates was particularly more effective than (2.8) and the value of F_{est} was required to prevent the possibility of overflow during the first iteration, (2.8) was used in all the algorithms compared in Section 7.

2.3 Beale's method

Although the function is guaranteed to decrease along the steepest-descent restart direction, the actual reduction is often poor compared with the reduction that would have occurred if restarting had not taken place. It would seem useful, therefore, if a cycle of n iterations could commence with the last direction of the

previous cycle. However, if an arbitrary vector is used as the initial direction in the computation of the sequence (2.1), then finite termination will not occur on quadratic functions because the vectors $\{p_k\}$ are not mutually conjugate. Beale (1972) has shown that, given an arbitrary initial direction p_0, the sequence of vectors

$$p_{k+1} = -g_{k+1} + \beta_k p_k + \gamma_k p_0,$$

where $\beta_k = y_k^T g_{k+1}/y_k^T p_k$ and $\gamma_k = y_0^T g_{k+1}/y_0^T p_0$, are mutually conjugate. The simple extension of this formula to nonlinear problems involves computing a cycle of n directions

$$p_{k+1} = -g_{k+1} + \beta_k p_k + \gamma_k p_t, \tag{2.9}$$

where $\beta_k = y_k^T g_{k+1}/y_k^T p_k$ and $\gamma_k = y_t^T g_{k+1}/y_t^T p_t$. The direction p_t is known as the *restart* direction and is the last direction of the previous cycle along which a linear search was made. Better performance will be obtained if the coefficients β_k and γ_k are computed as the solution of the linear system

$$\begin{pmatrix} y_k^T p_k & y_k^T p_t \\ y_t^T p_k & y_t^T p_t \end{pmatrix} \begin{pmatrix} \beta_k \\ \gamma_k \end{pmatrix} = \begin{pmatrix} y_k^T g_{k+1} \\ y_t^T g_{k+1} \end{pmatrix}. \tag{2.10}$$

This ensures that p_{k+1} is orthogonal to both y_k and y_t even when $F(x)$ is not quadratic. These equations are solved by a single back substitution since $y_t^T p_k$ is fixed at zero from the choice of β_{k-1} and γ_{k-1}. At the k-th iteration, (2.9) and (2.10) are used to compute p_{k+1} unless better progress can be made by using (2.1). If Beale's formula is considered unsatisfactory, a new cycle commences with p_k as the restart direction and with p_{k+1} computed from (2.1). Note that if (2.9) is used in a nonlinear conjugate-gradient algorithm, p_{k+1} may not be a descent direction, even if an exact linear search is made.

Powell (1977) has suggested a condition for restarting based upon the property that the gradient vectors are mutually orthogonal in the minimization of a quadratic function. If the gradient vectors are not sufficiently orthogonal, then a restart is made. Specifically, a restart will take place if

$$|g_k^T g_{k+1}| \geq 0.2\|g_{k+1}\|_2^2, \tag{2.11}$$

or there have been n linear searches in this particular cycle. A restart will take place also if p_{k+1} is not sufficiently downhill, an adequate downhill direction being one that satisfies the inequalities

$$-1.2\|g_{k+1}\|_2^2 \leq g_{k+1}^T p_{k+1} \leq -0.8\|g_{k+1}\|_2^2.$$

Apart from the first direction, the steepest-descent direction is very rarely used; it must be used, however, if an upper bound upon the step length prevents the traditional conjugate-gradient direction from being a descent direction.

Beale's algorithm may be generalized to accept inexact linear searches by replacing the direction $\bar{p}_{k+1}$ in condition (c3) by the direction that would be used if a restart were made in the $(k+1)$-th iteration. Thus, although a linear search is made along a direction generated by (2.9) the condition (c3) is checked by means of a direction computed from (2.1). It is necessary to alter the test for p_{k+1} being sufficiently downhill to

$$-g_{k+1}^T p_{k+1} \geq \sigma \|p_{k+1}\|_2 \|g_{k+1}\|_2. \tag{2.12}$$

The reasons for this will become clearer in Section 5 when we discuss convergence of the algorithm.

3. Limited-memory quasi-Newton methods

Although we have shown how to implement the conjugate-gradient method with an inexact linear search, the resulting algorithm may not always behave exactly as we should like when bounds are present or the gradient is expensive to compute. For this reason we seek methods which give a descent direction under much milder restrictions upon the step length. In this section we consider a class of such methods.

First we need some background theory on quasi-Newton methods. A quasi-Newton method computes the direction of search as the solution of the set of equations

$$B_{k+1} p_{k+1} = -g_{k+1}, \tag{3.1}$$

where B_{k+1} is an approximation to the Hessian matrix. At each iteration the approximate Hessian is updated by a matrix of rank two. The most popular updating formula is the *BFGS* formula,

$$B_{k+1} = B_k - \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T, \tag{3.2}$$

where $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k = \alpha_k p_k$. (The reader should note that the notation used here differs slightly from that of the general literature on quasi-Newton methods. It is customary to order the k-th iteration of a quasi-Newton method so that p_k is computed first, with the approximate Hessian being updated last of all. In order to make the notation consistent with that of conjugate-gradient

methods our quasi-Newton scheme computes the direction for the next iteration after updating the approximate Hessian.)

There are many alternative updating formulae, but they all generally satisfy the *quasi-Newton condition*:

$$B_{k+1}s_k = y_k.$$

In this paper we shall mainly consider updating formulae from the Broyden one-parameter family

$$B_{k+1}^{\phi} = B_k^{\phi} - \frac{1}{s_k^T B_k^{\phi} s_k} B_k^{\phi} s_k s_k^T B_k^{\phi} + \frac{1}{y_k^T s_k} y_k y_k^T + \phi_k (s_k^T B_k^{\phi} s_k) w_k w_k^T,$$

where

$$w_k = \frac{1}{y_k^T s_k} y_k - \frac{1}{s_k^T B_k^{\phi} s_k} B_k^{\phi} s_k,$$

and ϕ_k is a scalar function of y_k and $B_k^{\phi} s_k$ (see Broyden, 1970). For all positive ϕ_k, it can be shown that B_{k+1}^{ϕ} will be positive definite if B_k^{ϕ} is positive definite and $y_k^T s_k$ is positive. The positive definiteness of B_{k+1}^{ϕ} ensures that p_{k+1} is a descent direction.

The quasi-Newton direction can also be computed by updating the inverse of the approximate Hessian, and computing

$$p_{k+1} = -H_{k+1} g_{k+1}. \tag{3.3}$$

For example, the updating formula for the approximate inverse which corresponds to the BFGS formula for the approximate Hessian matrix itself is given by

$$H_{k+1} = H_k - \frac{1}{y_k^T s_k}(H_k y_k s_k^T + s_k y_k^T H_k) + \frac{1}{y_k^T s_k}(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}) s_k s_k^T. \tag{3.4}$$

For each update of B_k^{ϕ} for the approximate Hessian, there is a corresponding update for the approximate inverse Hessian. In the following discussion we shall need to refer to different updating formulae without explicitly displaying them. Let

$$H_{k+1}^{\phi} = \Gamma_{\phi}(H_k^{\phi}, y_k, s_k)$$

denote the updating formula used for a particular value of ϕ.

Perry (1977) and Shanno (1978a) have considered methods based upon computing p_{k+1} as $-H_{k+1} g_{k+1}$ where H_{k+1} is a matrix obtained by updating the

identity matrix with a limited number of quasi-Newton corrections. The storage of an $n \times n$ matrix is avoided by storing only the vectors that define the rank-two corrections, and consequently Shanno calls such methods "memoryless quasi-Newton methods". Since the methods require non-negligible storage, which may be substantial on some occasions, we prefer the term *limited-memory quasi-Newton methods*. The precise method depends upon the number of updating vectors stored and the quasi-Newton updating formula used. For example, the direction obtained with the "one-step" limited-memory BFGS update is given by (3.3), using (3.4) with H_k equal to the identity matrix, *viz*

$$p_{k+1} = -g_{k+1} + \frac{1}{y_k^T s_k}(s_k^T g_{k+1} y_k + y_k^T g_{k+1} s_k) - \frac{s_k^T g_{k+1}}{y_k^T s_k}(1 + \frac{y_k^T y_k}{y_k^T s_k}) s_k. \tag{3.5}$$

Although the direction of search is computed as the product of a matrix and a vector, the matrix H_{k+1} is never computed explicitly.

It may be verified by inspection of (3.5) that if the one-step limited-memory BFGS formula is applied with an exact linear search then p_{k+1} is identical to the direction obtained from the conjugate-gradient method (all vectors multiplied by $s_k^T g_{k+1}$ vanish). Hence it is possible for the limited-memory quasi-Newton method to generate mutually conjugate directions, but only if an exact linear search is made. This is demonstrated as follows: if the quasi-Newton condition is written in terms of the inverse Hessian we have $s_k = H_{k+1} y_k$; consequently

$$\begin{aligned} y_k^T p_{k+1} &= y_k^T H_{k+1} g_{k+1} \\ &= s_k^T g_{k+1}. \end{aligned}$$

Thus orthogonality between y_k and p_{k+1} will occur only if $g_{k+1}^T s_k$ is zero, that is, if an exact linear search is made.

Before we discuss other methods related to the BFGS formula, we shall describe a general r-step formula (for more details see Nazareth and Nocedal, 1978). For an r-step formula, the matrix H_{k+1} is implicitly defined by r approximate Hessian matrices $U_1, U_2, \ldots, U_r$ such that

$$H_{k+1} = \Gamma_\phi(U_r, y_{\sigma_r}, s_{\sigma_r}),$$

where the subscript ϕ denotes any quasi-Newton updating formula, σ_r is one of the indices of the r pairs of vectors $\{y_j, s_j\}$ and each U_{j+1} is related to its predecessor by the rule

$$U_{j+1} = \Gamma_\phi(U_j, y_{\sigma_j}, s_{\sigma_j}),$$

with U_1 usually equal to the identity matrix. The r pairs of vectors $\{y_j, s_j\}$ can be selected from any of those defined in earlier iterations and any combination of quasi-Newton updates can be used. (Strictly, we should have a suffix j on the parameter ϕ, but we have attempted to keep the notation as simple as possible.) In this notation we can define two methods based on the BFGS formula:

The one-step BFGS formula

At each iteration define U_1 as the identity matrix and set

$$H_{k+1} = \Gamma_{BFGS}(U_1, y_k, s_k). \tag{3.6}$$

The two-step BFGS formula

At each iteration define U_1 as the identity matrix and set

$$\begin{aligned} U_2 &= \Gamma_{BFGS}(U_1, y_{k-1}, s_{k-1}), \\ H_{k+1} &= \Gamma_{BFGS}(U_2, y_k, s_k). \end{aligned} \tag{3.7}$$

If the vectors $\{U_j y_{o_j}\}$ and $\{s_{o_j}\}$ are known for $j = 1, \ldots, r$, then $H_{k+1} g_{k+1}$ can be computed by the sequence of linear combinations

$$\begin{aligned} U_2 g_{k+1} &= lc\{U_1 g_{k+1}, U_1 y_{o_1}, s_{o_1}\} \\ U_3 g_{k+1} &= lc\{U_2 g_{k+1}, U_2 y_{o_2}, s_{o_2}\} \\ &\vdots \quad \vdots \\ H_k g_{k+1} &= lc\{U_r g_{k+1}, U_r y_{o_r}, s_{o_r}\}. \end{aligned}$$

A set of r pairs of vectors $\{U_j y_{o_j}, s_{o_j}\}$ define a single approximate Hessian matrix H_{k+1}.

Nazareth (1979) suggests a limited-memory quasi-Newton method in which as many pairs of vectors $\{y_j, s_j\}$ are kept as storage will allow. Unfortunately, storing only y_j and s_j substantially increases the cost of computing the direction of search since an additional $r(r+1)/2$ linear combinations must be used to compute the vectors $U_j y_{o_j}$. This additional cost is negligible for small values of r, but care must be taken lest the work required to compute the direction of search should dominate the computing cost of a single iteration.

The justification for using limited-memory quasi-Newton formulae is that p_{k+1} is guaranteed to be a descent direction if all the inner products $y_j^T s_j$ are positive for all vectors y_j and s_j used in the updating formulae. These inner products will nearly always be positive if the step-length is computed by means of algorithm

QNSL. Suppose that conditions (c1) and (c2) are satisfied with r equal to zero. Condition (c1) implies that

$$0 \leq (g(x_k + \alpha_k p_k) - \eta g_k)^T p_k < y_k^T p_k,$$

which implies that $y_k^T s_k$ is positive since α_k is positive. There is a very slight chance that p_{k+1} will not be a descent direction if the enforcement of condition (c2) causes $y_k^T s_k$ to be negative. However, this must imply that, for a step length smaller than α^t, the directional derivative at x_{k+1} must be larger than at x_k. The function $F(x)$ would need to be highly irregular for this to occur. If p_{k+1} is not a descent direction and a full quasi-Newton method is being used, then the approximate Hessian is not updated during that iteration. For a one-step limited-memory quasi-Newton method, the same strategy suggests taking a steepest-descent step. For a multi-step formula, some of the updates will need to be ignored.

Restarts can be incorporated into the limited-memory quasi-Newton scheme by performing a two-step update using s_k, y_k and information from a restart: s_t and y_t. Shanno (1978a) has suggested the following algorithm:

$$\begin{aligned} U_2 &= \Theta(U_1, y_t, s_t), \\ H_{k+1} &= \Gamma_{BFGS}(U_2, y_k, s_k), \end{aligned} \tag{3.8}$$

where Θ denotes the self-scaled BFGS formula

$$\begin{aligned} H_{k+1} &= \Theta(H_k, y_k, s_k) \\ &= \gamma H_k - \gamma \frac{1}{y_k^T s_k}(H_k y_k s_k^T + s_k y_k^T H_k) \\ &\quad + \frac{1}{y_k^T s_k}(1 + \gamma \frac{y_k^T H_k y_k}{y_k^T s_k}) s_k s_k^T, \end{aligned}$$

with $\gamma = y_k^T s_k / y_k^T H_k y_k$. (See Oren, 1974 and Oren and Spedicato, 1976, for a discussion of the self-scaled BFGS formula.) This algorithm is one of those tested in Section 7.

4. Preconditioned conjugate-gradient methods

Preconditioning has been used for some time to aid the solution of linear equations by conjugate-gradient methods. The purpose of preconditioning is to alter the coefficient matrix of the problem so that its condition number is reduced. It is hoped that this action will reduce the number of iterations required. Preconditioning can also be regarded as a technique which capitalizes on any structure occurring

in the coefficient matrix, and it is from this viewpoint that we shall discuss the application of preconditioning to nonlinear problems.

Suppose we are minimizing a quadratic function $c^T x + \frac{1}{2} x^T Q x$ where the Hessian matrix Q can be written in the form

$$Q = M - N,$$

with M a positive-definite symmetric matrix which is easy to invert (for example, M might be diagonal). The procedure for computing the direction of search may be generalized as follows (see Concus *et al.*, 1976). Solve the linear equations

$$M z_{k+1} = -g_{k+1}$$

and set

$$p_{k+1} = z_{k+1} + \beta_k p_k, \tag{4.1}$$

where $\beta_k = -y_k^T z_{k+1} / y_k^T p_k$. As in the traditional conjugate-gradient algorithm, the directions $\{p_k\}$ are mutually conjugate with respect to the matrix Q, but the algorithm has the additional property that the vectors $\{z_k\}$ are conjugate with respect to the matrix M.

This algorithm may be extended to nonlinear problems by using a matrix M_k that varies from iteration to iteration; for example, Nazareth and Nocedal (1978) suggest using an r-step limited-memory approximate Hessian matrix for M_k^{-1}. If the Hessian matrix is sparse, we can form a complete approximation to the Hessian matrix by using finite differences (Gill and Murray, 1973; Curtis *et al.*, 1974), or a sparse analogue of any of the well known updating formulae (Toint, 1977; Shanno, 1978b). If we denote the approximate Hessian as B_{k+1} the iteration requires the solution of the sparse equations

$$B_{k+1} z_{k+1} = -g_{k+1}. \tag{4.2}$$

One of the most effective methods for solving the sparse positive-definite equations $Ax = b$ is to form the LDL^T factorization

$$A = LDL^T,$$

where L is a unit-lower triangular matrix and D is a diagonal matrix. The vector x may then be found by successively solving the triangular systems

$$Lv = b \quad \text{and} \quad L^T x = D^{-1} v.$$

Unfortunately, whatever the method used to approximate the sparse Hessian, B_{k+1} can not be guaranteed to be positive definite. If indefiniteness occurs, not only is the Cholesky algorithm numerically unstable but also p_{k+1} may not be a descent direction. We can remove both these difficulties by using the *modified* LDL^T *factorization* of the matrix B_{k+1}. The modified LDL^T factorization of a matrix A is such that

$$A = LDL^T - E,$$

where L and D are defined as before and E is a diagonal matrix which is zero if A is positive definite (see Gill and Murray, 1974a).

If there is little fill-in during the factorization it is not worthwhile computing the direction p_{k+1} since z_{k+1} is a perfectly adequate direction of descent (compare Equations 3.1 and 4.2). However, there are some problems for which the amount of fill-in is too large for the factors to be held in core. In these situations the modified LDL^T factorization can be computed so that any unwanted fill-in is ignored. In this case

$$A = LDL^T - \Xi,$$

where the matrix Ξ is *not* a diagonal matrix. Alternatively, if the Hessian matrix has some specific structure, such as a band structure, but there are some "complicating elements" which cause significant fill-in during the factorization process, these elements can be ignored. Techniques of this kind have been used successfully to precondition linear systems (see Meijerink and Van der Vorst, 1977; Munksgaard, 1979).

Techniques based upon matrix factorizations are specifically designed for sparse systems. Since we are primarily concerned with problems whose Hessian matrix cannot be stored, we shall not consider these methods further in this paper.

One technique that can be applied to general problems is a preconditioning based upon a diagonal scaling. If the direction of search is obtained from Equation (3.1) the quasi-Newton formulae from the Broyden class may be simplified so that the matrix B_k does not appear in the rank two correction; for example, the BFGS formula becomes

$$B_{k+1} = B_k + \frac{1}{g_k^T p_k} g_k g_k^T + \frac{1}{\alpha_k y_k^T p_k} y_k y_k^T. \tag{4.3}$$

This result implies that even if the off-diagonal elements of B_k are unknown, the *diagonal* elements can still be recurred. These diagonal elements may be used to precondition the conjugate-gradient method. Let γ_j and ψ_j denote the j-th elements of g_k and y_k respectively. If $\Delta_{k+1} = \text{diag}(\bar{\delta}_1, \ldots, \bar{\delta}_n)$ and $\Delta_k = \text{diag}(\delta_1, \ldots, \delta_n)$ denote the approximate diagonal Hessians during the $(k+1)$-th and k-th iterations

respectively, then

$$\bar{\delta}_j = \delta_j + \frac{1}{g_k^T p_k}\gamma_j^2 + \frac{1}{\alpha_k y_k^T p_k}\psi_j^2, \tag{4.4}$$

and

$$\Delta_{k+1} z_{k+1} = -g_{k+1}. \tag{4.5}$$

The BFGS recurrence of Δ_k has some theoretical justification since, in the quadratic case with exact linear searches, the search directions generated by the two algorithms are identical (see Nazareth, 1979). The motivation for using this algorithm on general nonlinear functions is to scale the search directions so that the initial step along p_k will be a better prediction of the minimum. We shall demonstrate in Section 7 that this is indeed the case.

To ensure that the elements of Δ_{k+1} are positive, a diagonal is not updated if the result will be negative. An algorithm must also be used to prevent the condition number of Δ_{k+1} from becoming excessive. *If* Ω denotes a preset bound on the condition number of Δ_{k+1} and $\kappa > \Omega$ where $\kappa = \bar{\delta}_{max}/\bar{\delta}_{min}$, then we use $\bar{\delta}_j^w$ for the new diagonals, where $w = \log\Omega/\log\kappa$. The value of the bound κ is the same as that used by Gill *et al.* (1972b), *viz*

$$\Omega = 1/(100n^{\frac{1}{2}}\epsilon),$$

where ϵ is the relative machine precision. It should be noted that this bound on the condition number was never achieved in any of the computer runs discussed in Section 7.

The traditional conjugate-gradient algorithm and Beale's algorithm preconditioned by a diagonal matrix still suffer from the disadvantage that the rate of convergence may deteriorate if there is a bound upon the step length. Fortunately, the limited-memory quasi-Newton methods can also be generalized to accept diagonal preconditioning. In the r-step formula, the sequence of r approximate Hessian matrices $U_1, \ldots, U_r$ is started with U_1 equal to a diagonal preconditioning matrix rather than the identity matrix. For example, the preconditioned two-step BFGS algorithm is given by:

The diagonally preconditioned two-step BFGS formula

At each iteration define U_1 as the diagonal matrix Δ_{k+1}^{-1} and set

$$\begin{aligned} U_2 &= \Gamma_{BFGS}(U_1, y_{k-1}, s_{k-1}), \\ H_{k+1} &= \Gamma_{BFGS}(U_2, y_k, s_k). \end{aligned} \tag{4.6}$$

5. Some comments on convergence proofs

It is important to realize that all proofs concerning the convergence of conjugate-gradient algorithms are theoretical in the sense that they ignore the effects of rounding error. Rounding error may upset some of the most fundamental theory of conjugate-gradient algorithms, as the following example will illustrate.

Consider the quadratic function

$$F(x) = \frac{1}{2}\sum_{j=1}^{n} d_j(1 - x_j)^2,$$

where $d_j = (j/n)^p$, $p > 0$. For all values of n and p this function has the solution $x^* = (1, \ldots, 1)^T$, at which point $F(x)$ is zero. The Hessian matrix of this function is of the form $Q = \text{diag}((1/n)^p, (2/n)^p, \ldots, 1)$ and has condition number n^p. For $p = 3$ and $n = 50$ the condition number of Q is 1.25×10^5. This case was run on an IBM 370/168 using double-precision arithmetic, the relative machine precision being approximately 1.0×10^{-15}. In theory, the conjugate-gradient algorithm should give zero values of $F(x)$ and $\|g(x)\|$ at iteration n. However, the function and gradient values were 9.8×10^{-5} and 9.8×10^{-4} respectively. At iteration $2n$ the function value and gradient norm were 1.3×10^{-8} and 8.0×10^{-5}. Clearly the property of n-step termination does not apply in this case, yet the condition number of Q is not excessive in relation to the precision of the arithmetic being used.

It is illuminating to compare this result with that obtained by another method which theoretically should compute exactly the same numbers when minimizing a quadratic function. Nazareth (1979) has shown that on a quadratic function with exact linear searches, the search directions generated by the BFGS method are identical to those generated by the traditional conjugate-gradient method. If we use the BFGS formula to minimize the quadratic function (2.5) we find that at the 50th iteration the function value is 2.1×10^{-5} and the norm of the gradient vector is 8.8×10^{-5}. However, at iteration 55 the method recovers and the function value is essentially zero at 4.1×10^{-28} with gradient norm 1.5×10^{-14}.

These results indicate that in practice, even on a quadratic function, the sequence generated by a conjugate-gradient algorithm will be infinite. Consequently, we must not be surprised if behavior predicted by a theoretical result based upon some n-step property is not observed in practice. The most useful theorems are those which provide *negative* results, that is, results which tell us that convergence will not occur, even if infinite-precision arithmetic is used. For example, one such result was established by Powell (1976): if the initial direction of search for a

sequence of iterates designed to minimize a quadratic function is not the steepest-descent direction then the traditional conjugate-gradient method will converge only at a linear rate.

The most powerful convergence proof for the traditional conjugate-gradient algorithm with exact linear searches was established by Powell (1977). However, for Powell's proof to be correct, it is necessary to modify the statement of the theorem.

Theorem 1 (Convergence with exact linear searches).

Let $\{x_k\}$ and $\{p_k\}$ be computed from the formulae (2.1) and (2.3) for all k. Choose each α_k such that it is the local minimum along p_k which is the smallest in magnitude. If the set $\{ x \mid F(x) \leq F(x_0) \}$ is bounded, if $g(x)$ is continuous and if the quantities $\|x_{k+1} - x_k\|_2$ tend to zero, then

$$\lim_{k \to \infty} \|g_k\| = 0. \quad \blacksquare$$

Our specification of the sequence differs from that of Powell in the requirement that each α_k be the local minimum along p_k that is closest to x_k. This must be done to ensure that members of the sequence $\{\alpha_k\}$ satisfy the condition (2.5) (see Ortega and Rheinbolt, 1970, pp. 483-484). If the sufficient decrease is not made, the proof given by Powell does not hold since (2.5) is needed to show that $\lim_{k \to \infty} g_k^T p_k / \|p_k\| = 0$ and hence that $\lim_{k \to \infty} \|g_k\| = 0$. The unmodified theorem breaks down under the following circumstances.

Let α_k^* be the minimum of $F(x_k + \alpha p_k)$ closest to zero. Suppose that the initial member of the sequence $\{\alpha^j\}$ lies beyond α_k^*. If $F(x)$ is a non-unimodal function it is possible for $g(x_k + \alpha^0 p_k)^T p_k$ to be negative and for $F(x_k + \alpha^0 p_k)$ to be less than F_k. The step-length algorithm will then proceed to locate a value of α_k that is greater than α_k^*. When this occurs the function may be reduced by an arbitrarily small amount and it is not possible to derive a condition analogous to (c2). In practical computation it is relatively simple to derive an acceptable step length from any local minimum of $F(x_k + \alpha p_k)$ by simply reducing the step by multiples of a constant less than unity. Unfortunately the resulting step length does not give the required orthogonality between g_{k+1} and p_k, which is an essential feature of Powell's analysis.

It must be emphasized that the need to determine the first minimum along p_k renders the algorithm analysed in Theorem 1 unsuitable for computer implementation. If we are to be sure that α_k is the first minimum along p_k for general functions, we need to be able to compute *all* the minima of $F(x_k + \alpha p_k)$. Nonetheless, Theorem 1 does have practical relevance for convex functions or functions that are unimodal along each direction p_k.

We shall now present a convergence theorem for the algorithm with inexact linear searches, which is analogous to Theorem 1.

Theorem 2 (Convergence with inexact linear searches).

Let p_k and β_k be computed as in Theorem 1. Let α_k be computed by means of the step-length algorithm TCGSL. If α_k is computed as the first minimum along p_k only on those occasions that the condition (c3) is not satisfied, then

$$\lim_{k\to\infty} \|g_k\| = 0.$$

Proof

Gill and Murray (1974c) have shown that the step-length algorithm QNSL gives an α_k such that (2.5) is true. Consider the step lengths generated by algorithm TCGSL. If a value of α^t is found that eventually satisfies (c3), then α_k will satisfy (2.5) since it can be regarded as being computed by means of QNSL but with modified values of η and μ. Alternatively, if the minimum of $F(x_k + \alpha p_k)$ is found, then (2.5) is seen to be true from the analysis presented by Ortega and Rheinbolt (1970, pp. 484-486).

Assume that the theorem is not true and that a limit point is obtained which is not a stationary point of $F(x)$. We can choose an integer m and a small scalar ϵ such that for all $k \geq m$,

$$\|g_k\| \geq \frac{\epsilon}{\sigma}. \tag{5.1}$$

Since the quantities $\|x_{k+1} - x_k\|$ tend to zero we must have $\lim_{k\to\infty} F_k - F_{k+1} = 0$, which implies from (2.6) that for m sufficiently large,

$$-\frac{g_k^T p_k}{\|p_k\|} < \epsilon.$$

Thus from (5.1),

$$-\frac{g_k^T p_k}{\|g_k\|\|p_k\|} < \sigma$$

for all $k \geq m$. However this implies that α_k will be computed with an exact linear search and the algorithm will behave precisely as the conjugate-gradient algorithm discussed in Theorem 1. Since such an algorithm is convergent we have a contradiction and the theorem must be true. ∎

McCormick and Pearson (1969) have shown that for a wide class of functions,

the restarted conjugate-gradient method is n-step superlinearly convergent, i.e.

$$\lim_{j\to\infty} \frac{\|x_{nj+n} - x^*\|}{\|x_{nj} - x^*\|} = 0.$$

The proof of n-step superlinear convergence is critically dependent upon the use of restarting. Powell (1976) has shown that algorithms that do not contain a restarting strategy almost always converge linearly.

The reader should note that the term n-step superlinear convergence has very little meaning when n is large since the computation of enough values for the asymptotic convergence theory to hold would be infeasible. In our view, any conjugate-gradient method that requires more than $2n$ or $3n$ iterations to achieve any meaningful accuracy should be considered to have failed. Conjugate-gradient methods are intended for values of n that may be in excess of 1000 (for example, in molecular chemistry, problems with several thousand variables are routine). Computing several thousand values of the objective function is likely to be prohibitively expensive for all but the simplest of problems.

In summary, our experience is that, except in very special circumstances (such as when $F(x)$ is a well-conditioned quadratic function), the conjugate-gradient method is *always* linearly convergent, regardless of whether or not restarting takes place.

6. Restarting strategies and extensions

In our view, the accepted theoretical justification for restarting in the nonlinear case needs to be re-examined. In particular, restarting every n iterations would appear to be unnecessary since we should not expect to perform more than $2n$ or $3n$ iterations in total. It is unlikely that one or two iterations with the steepest-descent direction will make much difference to the progress of the minimization.

It would appear that the same argument could be used to dismiss the case for restarting with arbitrary directions, as is done in Beale's algorithm. However, the results of Section 7 show that, on average, Beale's algorithm with Powell restarts generally requires fewer iterations and function evaluations than the traditional conjugate-gradient method. Initially we found this puzzling since it seemed unlikely that a direction, kept for what could be a cycle of thirty or forty iterations, could possibly provide any useful information. However, this question was answered when the runs were closely investigated. On many of the problems a restart was being made every one or two iterations. It was extremely rare for a direction to be used for more than ten iterations. Clearly in these cases, the term "restarting"

is misleading. The fact that Powell's restart criterion causes frequent restarts is not surprising, since a criterion based upon the orthogonality of the gradient vectors will have no meaning when inexact linear searches are made (Powell's paper considered the conjugate-gradient method with exact linear searches).

We believe that the major contributing factor to the improvements obtained by restarting is the additional information that is being used during the computation of the search direction, namely, the restart direction p_t and its corresponding gradient difference y_t. In the quadratic case the conjugate-gradient direction p_{k+1} of (2.1) is essentially of the form

$$p_{k+1} = -g_{k+1} + \sum_{j=0}^{k} \omega_j p_j, \tag{6.1}$$

with $\omega_k = \beta_k$ and $\omega_j = 0$ for $0 \leq j < k$. In the nonlinear case, all the ω_j will be nonzero and Beale's algorithm can be interpreted as a normal conjugate-gradient iteration with a recent direction being maintained in the recurrence.

However, there are some aspects of restarting that remain inexplicable. In a numerical experiment, two implementations of Shanno's method were tested on the examples listed in Section 7.3. The methods were identical except that one algorithm was restarted every iteration while the other was restarted according to Powell's criterion. Although the number of function evaluations required was of the same order of magnitude for both techniques, the algorithm which was restarted every iteration required more evaluations, on average, than its competitor, despite the fact that the Powell criterion forces a restart very frequently. In our opinion this behaviour is not adequately explained by the currently accepted theory.

It is clear that we need to be able to compute a direction incorporating information from past iterations without significantly increasing the storage. We shall now describe a technique that we have found to be quite successful in practice. After a sequence of iterations of a conjugate-gradient method we could pretend that we had obtained the current point by taking just one step along the vector representing the total change in the variables. We would expect this vector to be a good direction of descent since a substantial decrease in $F(x)$ must have already occurred. A feature common to all conjugate-gradient type methods is that eventually very small steps are taken with very little reduction in the function value. We can regard the computation of the total change in x as a method of identifying the "trend" of a sequence of smaller steps. We can utilize these observations by modifying any of the conjugate-gradient methods so that a cycle of iterations is made with the recurrence relation including a term of the form

$$\bar{s}_k = \sum_{j=t}^{k-1} s_j = x_k - x_t, \tag{6.2}$$

where t is the index of the iteration in which the current cycle commenced. The corresponding change in the gradient is

$$\bar{y}_k = g_k - g_t = \sum_{j=t}^{k-1} y_j. \tag{6.3}$$

A new cycle should start when the reduction in $F(x)$ begins to become small in relation to the total reduction that has occurred since the start of the current cycle. One method of achieving this objective is as follows.

Let θ_q be a scalar which is constant during the q-th cycle; a new cycle starts at iteration k (i.e. t is set equal to k) if

$$F_k - F_{k+1} \leq \theta_q (F_{t+1} - F_{k+1}), \tag{6.4}$$

where F_{t+1} is the value of the function on completion of the first iteration of cycle q. The parameter θ_{q+1} will be larger or smaller than θ_q according to whether or not $F_t - F_{t+1}$, the reduction in $F(x)$ during the last iteration of cycle q, is larger or smaller than $F_{t+1} - F_{t+2}$, the reduction obtained during the first iteration of cycle $q+1$. The precise method for fixing θ_{q+1} is as follows: initially, $\theta_0 = 10^{-2}$; after the first iteration of the $(q+1)$-th cycle which started at iteration t:

$$\theta_{q+1} = \begin{cases} 2\theta_q, & \text{if } F_t - F_{t+1} \leq \frac{1}{2}(F_{t+1} - F_{t+2}); \\ \frac{1}{2}\theta_q, & \text{if } F_t - F_{t+1} > 2(F_{t+1} - F_{t+2}). \end{cases}$$

In Section 7 we describe an implementation of this scheme with a preconditioned two-step BFGS formula. The precise algorithm is as follows:

The diagonally preconditioned two-step BFGS formula with accumulated step

At each iteration define U_1 as the diagonal matrix Δ_{k+1}^{-1} and set

$$\begin{aligned} U_2 &= \Gamma_{BFGS}(U_1, \bar{y}_k, \bar{s}_k); \\ H_{k+1} &= \Gamma_{BFGS}(U_2, y_k, s_k). \end{aligned} \tag{6.5}$$

7. Numerical results and discussion

In this section we discuss the numerical behaviour of several of the methods discussed in earlier sections. It was not feasible to test every technique that has been described, but we have attempted to select those algorithms which are either in common use or show the most promise from a theoretical point of view. The method used to compare algorithms consists of applying them to a set of test problems. We do not claim that this is a completely satisfactory means of comparison, but we believe that, if the test problems are selected carefully, the evidence obtained can be a valuable aid in the selection of the best algorithm.

7.1 The assessment criterion

All optimization software requires a criterion for terminating the computation of the sequence $\{x_k\}$. Ideally, if we wish to measure the comparative efficiency of routines we should set the same termination criterion in all the routines tested and then compute the cost of a minimization, in terms of the number of function evaluations for instance. However, there is no universal agreement on what is the best termination criterion and a different criterion used by another researcher may result in a wide variation in the accuracy of the answer obtained. The question remains, therefore, as to the point at which we should assess the efficiency of the various methods. The assessment criterion used here is to take the first point x_k for which

$$F_k - F(x^*) < \tau(1 + |F(x^*)|), \tag{7.1}$$

where τ is a scalar. Some authors have argued against the use of (7.1) because it includes $F(x^*)$, which is unknown on real problems. We believe that such authors are confusing an *assessment criterion*, where the use of $F(x^*)$ is legitimate, with a *termination criterion*, where it is not.

If the criterion (7.1) is to give a realistic *assessment* of the performance of an algorithm, the choice of τ must give a point x_k which is close to a final estimate of x^* obtained with a realistic *termination* criterion. The relative performance of algorithms with superlinear convergence is almost invariant with the choice of τ and a very small value can be used. For example, on an IBM 370/168, where the function can be computed to approximately fifteen decimal places in double precision, a reasonable choice of τ is 10^{-10}. However, for conjugate-gradient type methods, which exhibit a linear rate of convergence, the performance can vary widely with the choice of τ. It is not unusual for the number of function evaluations to be three times greater for $\tau = 10^{-10}$ than for $\tau = 10^{-5}$. In this case *it is important that a moderate termination criterion be used.* In all the tests carried

out for this study, τ was set at 10^{-5}.

7.2 The algorithms tested

The results of Section 7.5 illustrate the numerical behaviour of eight algorithms for large-scale unconstrained minimization.

The following four conjugate-gradient algorithms all incorporate the step-length algorithm TCGSL.

Algorithm CG

The traditional conjugate-gradient algorithm (see 2.1, 2.3 and 2.8).

Algorithm PCG

Diagonally preconditioned traditional conjugate-gradient algorithm (see 2.1, 2.3, 4.1, 4.4 and 4.5).

Algorithm BCG

Beale's algorithm with the Powell restart criterion (see 2.8, 2.9, 2.10, 2.11 and 2.12).

Algorithm PBCG

Algorithm BCG with diagonal preconditioning.

The following four limited-memory quasi-Newton algorithms all incorporate step-length algorithm QNSL.

Algorithm Shanno

An implementation of Shanno's algorithm (see 3.8 and Shanno, 1978a). Apart from the use of (3.8), we have attempted to follow the description of Shanno's algorithm as closely as possible. However, we must emphasise that the results obtained will not necessarily agree with those of other implementations.

Algorithm PLM1

Diagonally preconditioned one-step BFGS formula (see 3.6). The direction of search is computed as $-H_{k+1}g_{k+1}$ where H_{k+1} is given by

$$H_{k+1} = \Gamma_{BFGS}\left(\Delta_{k+1}^{-1}, y_k, s_k\right).$$

Algorithm PLM2

Diagonally preconditioned two-step BFGS formula (see 3.7 and 4.6).

Algorithm PLMA

Diagonally preconditioned two-step BFGS formula with accumulated step (see 6.5, 6.2, 6.3 and 6.4).

For comparison purposes, the smaller test problems were run on two algorithms designed to solve small dense problems.

Algorithm MNM

A modified Newton method using first and second derivatives (see Gill and Murray, 1974a).

Algorithm QNM

A quasi-Newton method using the full $n \times n$ BFGS update of the approximate Hessian Matrix (see Gill and Murray, 1972a).

7.3 The test examples

The provision of suitable test problems is extremely difficult. Problems that are used to measure the efficiency of algorithms for small dense problems are completely unsatisfactory since the algorithms considered here are intended only for large-scale problems. For example, it is pointless to test a two-step limited memory quasi-Newton method on a two dimensional problem since the algorithm will be effectively performing a *full quasi-Newton iteration!*

A serious difficulty with using very large test problems is that, for all but the most trivial examples, the CPU time necessary to compute the objective function will be very large. This is typically the case if we attempt to use real-world problems for testing purposes. Moreover, it is desirable that problems be defined in such a way that they may be used by other researchers. Large-scale real-world problems almost invariably are written in a non-portable form or can be run only with vast quantities of numerical data.

In this study we have attempted to compromise on these issues by collecting a set of non-trivial problems that can be run with moderate ease at other installations. A total of 25 problems are considered. Of these, 20 problems are of dimension 50 or greater and 9 problems are of dimension 100. For the purposes of the comparison, each algorithm being tested was run a total of 93 times. It is necessary to present an extensive number of results because, as we shall demonstrate in Section 7.5, the performance of conjugate-gradient methods is generally erratic. If we are to identify which strategy gives a true improvement in performance, a wide spectrum of results must be considered.

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. Pen1 (Gill *et al.*, 1972b)

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

The solution varies with n, but $x_i = x_{i+1}$, $i = 1, \ldots, n-1$. All the runs made were 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. Pen2 (Gill *et al.*, 1972b)

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

where $c_i = e^{i/10} + e^{(i-1)/10}$ for $i = 2, \ldots, n$. The solution varies with n, but $x_i = x_{i+1}$ for $i = 1, \ldots, n-1$. 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. Pen3

$$F(x) = a\Bigg\{1 + e^{x_n}\sum_{i=1}^{n-2}(x_i + 2x_{i+1} + 10x_{i+2} - 1)^2$$

$$+\Big(\sum_{i=1}^{n-2}(x_i + 2x_{i+1} + 10x_{i+2} - 1)^2\Big)\Big(\sum_{i=1}^{n-2}(2x_i + x_{i+1} - 3)^2\Big)$$

$$+ e^{x_n - 1}\sum_{i=1}^{n-2}(2x_i + x_{i+1} - 3)^2\Bigg\}$$

$$+\Big(\sum_{i=1}^{n}(x_i^2 - n)\Big)^2 + \sum_{i=1}^{n}(x_i - 1)^2.$$

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

Example 4. PSP (Toint, 1978)

$$F(x) = \sum_{i=1}^{50}\alpha_i(x_i - 5)^2 + \sum_{i=1}^{33}\beta_i h_i(y_i),$$

where

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

$$h_i(y_i) = \begin{cases} 1/y_i, & y_i \geq 0.1, \\ 100(0.1 - y_i) + 10, & y_i < 0.1, \end{cases}$$

and the constants α_i, β_i and d_i are given by Toint (1978) together with the sets of indices $A(i)$ and $B(i)$ which are subsets of the indices $\{1, 2, 3, \ldots, 50\}$. This example has a sparse Hessian matrix.

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

Example 5. Chebyquad (Fletcher, 1965)

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

where

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

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

Example 6. Watson (Brent, 1973)

$$F(x) = \sum_{i=2}^{30}\left(\sum_{j=2}^{n} x_j(j-1)\left(\frac{i-1}{29}\right)^{j-2} - \left(\sum_{j=1}^{n} x_j\left(\frac{i-1}{29}\right)^{j-1}\right)^2 - 1\right)^2 + x_1^2 + \left(x_2 - x_1^2 - 1\right)^2.$$

This problem was included to demonstrate the poor convergence of conjugate-gradient type methods on mildly ill-conditioned problems whose eigenvalues are *uniformly distributed*. Only the value $n = 6$ was tested, the condition number of the Hessian matrix being approximately 8.6×10^4.

Example 7. GenRose

This function is a generalization of the well-known two-dimensional Rosenbrock function (Rosenbrock, 1960).

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

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 (for a more detailed discussion of this point, the reader is referred to Gill and Murray, 1979).

The next four examples arise from the discretization of problems in the calculus of variations. Similar problems arise in 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 tri-diagonal Hessian matrix.

Example 8. Calvar1 (Gill and Murray, 1973)

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

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

Example 9. Calvar2

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

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

Example 10. Calvar3 (Gill and Murray, 1973)

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

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

Example 11. Var(λ) (Toint, 1978)

$$J(x(t)) = \int_0^1 \left\{ x'(t)^2 + 2\lambda e^{x(t)} \right\} dt,$$

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

This function is discretized using piecewise linear functions instead of piecewise cubics. Like the other discretized problems, the Hessian is tri-diagonal. The problem is quadratic for $\lambda = 0$.

Example 12. QOR (Toint, 1978)

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

where the constants a_i, β_i, d_i and sets $A(i)$ and $B(i)$ are the same as those used in example PSP. This function is convex with a sparse Hessian matrix.

Example 13. GOR (Toint, 1978)

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

where

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

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

and

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

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

Example 14. ChnRose (Toint, 1978)

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

where the constants a_i are those used in the example PSP. The value of $F(x)$ at the solution has been modified as in Example 7. The Hessian matrix is tri-diagonal.

7.4 Starting points

The starting points used were the following.

Start 1

$$x_0 = (0, 0, \ldots, 0)^T.$$

Start 2

$$x_0 = \left(\frac{1}{n+1}, \frac{2}{n+1}, \ldots, \frac{n}{n+1}\right)^T.$$

Start 3

$$x_0 = (1, -1, 1, -1, \ldots)^T.$$

Start 4

$$x_0 = \left(0, -\frac{.2}{n+1}, -\frac{.6}{n+1}, \ldots, (1-n)\frac{.1n}{n+1}\right)^T.$$

Start 5

$$x_0 = \left(\frac{1}{2}, \frac{1}{2}, \ldots \frac{1}{2}\right)^T.$$

Start 6

$$x_0 = (-1, -1, \ldots, -1)^T.$$

7.5 Discussion of the results

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

Each problem was solved using three values of η, the step-length accuracy. These values were 0.25, 0.1 and 0.001. Each algorithm requires two additional user-specified parameters: λ, the bound upon the change in x at each iteration, (the quantity $\|x_{k+1} - x_k\|_2$) and F_{est}, an estimate of the value of the objective function at the solution. The value of λ was set at 10^5 for all problems except Pen1, Pen2 and Chebyquad where it was set at 10 to avoid overflow during the computation of the objective function. In every case F_{est} was set to the value of $F(x)$ at the solution.

The full set of results are contained in Tables A1–A8 of the appendix. Table A1 contains the number of function evaluations required by conjugate-gradient type methods on problems whose Hessian matrices have clustered eigenvalues. Table A2 gives equivalent results for limited-memory quasi-Newton methods. Tables A3 and A4 give the number of function evaluations required by conjugate-gradient methods and limited-memory quasi-Newton methods on general problems.

In order to limit the computing costs, an upper bound was placed upon the number of function evaluations made during each minimization. In all cases where

this bound was achieved the function was still being reduced (albeit slowly!). The bound varied from problem to problem, but in most cases where the bound was achieved, the progress of the minimization was sufficiently slow that considerably more function evaluations would have been required to obtain any meaningful accuracy. An upper limit of 2000 function evaluations was placed on all problems except Watson, for which the limit was 700. The large value of the limit (relative to the size of n) for Watson reflected the fact that the objective function is inexpensive to compute. Two algorithms, PBCG and PLMA, successfully solved all the test problems.

It is helpful if the results of the appendix are summarized so that the methods may be easily "compared". Tables 1 and 2 contain the total number of function evaluations required by each method on the two classes of test example. Clearly the reader should be wary of using these tables as the only means of comparison since the totals will depend upon the η-values used and the limit on the total number of function evaluations. However, we believe that Tables 1 and 2 serve as a useful introduction to the tables given in the appendix.

The results indicate that the performance of conjugate-gradient type methods may be erratic. For example, quasi-Newton methods almost invariably have the property that the number of iterations decreases monotonically as the step-length parameter η is reduced to zero. This implies that the "best" value of η determined by averaging the results for a large set of test problems will be close to the "best" η for each individual problem. However, for conjugate-gradient type methods the variation in the number of iterations as η changes is often far from uniform. (In this case a user may find it worthwhile to experiment with the choice of η if a large number of similar problems are being solved.)

Table 1
Total Number of Function Evaluations Required
to Solve Problems With Clustered Eigenvalues

METHOD	EVALUATIONS
CG	2189
BCG	1922
PCG	2269
PBCG	2177
Shanno	1858
PLM1	2062
PLM2	2085
PLMA	1948

This erratic behavior makes it very difficult to draw firm conclusions about whether one algorithm is any better than another. However, there is one aspect concerning the implementation of the methods on which a firm statement can be made. The implementations of all the algorithms discussed in this report incorporate a sophisticated routine to compute the step length. We believe that such a routine is vital for both efficiency and robustness.

The results of Tables A1–A4 indicate that the traditional conjugate-gradient method and Beale's method are very effective for problems whose Hessian matrix has clustered eigenvalues. However, these methods may require a prohibitively large number of function evaluations for general problems. We believe that the reputation of the traditional conjugate-gradient method for unreliability partly stems from the use of an inadequate step-length routine. The results reported here are quite acceptable but may represent the best that can be achieved with an unmodified algorithm.

We would expect preconditioning to have a negative effect upon the rate of convergence for problems whose Hessian matrix already has sets of clustered eigenvalues. However, the results of Table 1 show that the degradation in performance on "naturally" preconditioned problems is not serious. Moreover, as Table 2 shows, the improvement in performance on general problems is quite dramatic overall. For this reason we believe diagonal preconditioning to be well worthwhile.

Table 2
Total Number of Function Evaluations Required
to Solve General Problems

METHOD	EVALUATIONS
CG	32820
BCG	28588
PCG	16597
PBCG	14714
Shanno	24434
PLM1	17188
PLM2	16934
PLMA	13985

A feature of methods which use preconditioning is that the property of n-step termination is lost. (In Section 5 this property was shown to hold numerically only if the Hessian matrix is well-conditioned.) A comparison of the results for the well-conditioned quadratic function Var(0) shows that preconditioning may result in three times the number of function evaluations required by an algorithm that

has n-step termination. However, this ratio is reduced significantly when the near-quadratic function Var(1) is minimized.

We were surprised that the two-step BFGS formula appeared to give little advantage over the one-step BFGS formula. For this reason it is not clear that increasing the number of updates will necessarily improve the performance of a limited-memory quasi-Newton method. This point is emphasised on the numerous occasions that the limited-memory method PLMA out-performs the full quasi-Newton method.

The most successful method was PLMA, in terms of the total number of function evaluations required to solve this set of test problems. In a direct comparison with any single alternative routine, the number of problems on which PLMA was more successful was noticeably greater than the number on which it was less successful. Thus the additional storage required to implement a more sophisticated routine seems to be warranted on the grounds of both improved efficiency and robustness.

Tables A5 and A6 give a comparison of function evaluations for PLMA and the two methods designed for dense problems. These results are summarized in Tables 3 and 4. Tables A7 and A8 give a comparison of *iterations* for the same problems. (The reader should note that the values of η used for these comparisons are not optimal for the methods MNM and QNM.)

Table 3
Total Number of Function Evaluations Required
to Solve Problems With Clustered Eigenvalues

METHOD	EVALUATIONS
PLMA	703
QNM	1738
MNM	274

Table 4
Total Number of Function Evaluations Required
to Solve General Problems

METHOD	EVALUATIONS
PLMA	5505
QNM	3786
MNM	1819

The comparison of PLMA with MNM and QNM was not intended to be an exhaustive one, but merely to demonstrate the efficiency of methods specifically designed for large-scale problems compared to general methods. As we might expect, on problems with clustered eigenvalues PLMA does relatively well, especially compared to QNM. The overall performance of PLMA compared to that of QNM is also respectable. Perhaps the most noticeable feature of this set of results is the often spectacular performance of the modified Newton method MNM.

8. Summary

An algorithm for large-scale optimization based upon preconditioning a two-step BFGS limited-memory quasi-Newton method has been suggested. An implementation of the method has been shown to be efficient on a selection of large test problems. We have shown that the performance of this and other conjugate-gradient algorithms varies significantly with the accuracy to which the step length is computed and the type of problem being solved. This suggests that a user should experiment with the step-length parameter when solving many similar problems.

A key point when solving problems by a conjugate-gradient algorithm is not to attempt to find too accurate a solution. The rate of convergence for all the methods is effectively linear. On a machine with a t decimal digit mantissa, we suggest termination when $F(x_k)$ is estimated to have, at most, $\min\{t/2, 5\}$ digits of accuracy. For many applications this approximate solution is more than adequate, but the low level of accuracy may imply that we are unable to determine that a correct solution has been found.

In order to solve very large problems, computing restrictions may require the number of iterations to be a small multiple of n. It should be noted that, even with the optimal value of the step-length accuracy, the method judged to be the best of those tested often failed to achieve this objective.

Acknowledgements

The authors wish to thank Margaret Wright, Michael Saunders and Enid Long for their careful reading of the earlier drafts of this report and a number of helpful comments. The authors also acknowledge the help of Mary Fenelon, Nick Gould and Mukund Thapa in obtaining some of the numerical results.

APPENDIX

TABLE A1
Number of *Function Evaluations* Required
by Conjugate-gradient Type Methods
on Problems With Clustered Eigenvalues

PROBLEM	η	CG	BCG	PCG	PBCG
Pen1 Start 3 $n = 50$	$\eta = .25$	37	46	44	44
	$\eta = .1$	27	27	31	31
	$\eta = .001$	32	32	38	38
Pen1 Start 3 $n = 100$	$\eta = .25$	40	40	42	42
	$\eta = .1$	9	9	11	11
	$\eta = .001$	9	9	11	11
Pen1 Start 2 $n = 50$	$\eta = .25$	19	19	82	81
	$\eta = .1$	25	32	54	57
	$\eta = .001$	33	33	62	50
Pen1 Start 2 $n = 100$	$\eta = .25$	17	17	62	67
	$\eta = .1$	45	45	84	72
	$\eta = .001$	58	58	117	125
Pen2 Start 5 $n = 50$	$\eta = .25$	22	22	43	58
	$\eta = .1$	24	24	45	44
	$\eta = .001$	44	46	67	124
Pen2 Start 5 $n = 100$	$\eta = .25$	15	15	11	11
	$\eta = .1$	13	13	12	12
	$\eta = .001$	21	19	18	18

KEY

CG - traditional conjugate-gradient method.
PCG - algorithm CG with diagonal preconditioning.
BCG - Beale's method with Powell restarts.
PBCG - diagonally preconditioned Beale's method.

TABLE A1 (continued)
Number of *Function Evaluations* Required
by Conjugate-gradient Type Methods
on Problems With Clustered Eigenvalues

PROBLEM	η	CG	BCG	PCG	PBCG
Pen2 Start 3 $n = 50$	$\eta = .25$	99	105	52	65
	$\eta = .1$	90	94	74	74
	$\eta = .001$	121	121	106	106
Pen2 Start 3 $n = 100$	$\eta = .25$	17	20	14	14
	$\eta = .1$	13	19	17	17
	$\eta = .001$	28	32	30	30
Pen3 Start 1 $n = 50$	$\eta = .25$	97	79	64	71
	$\eta = .1$	94	84	77	57
	$\eta = .001$	119	89	94	73
Pen3 Start 1 $n = 100$	$\eta = .25$	105	77	70	74
	$\eta = .1$	108	74	76	74
	$\eta = .001$	133	92	132	99
Pen3 Start 3 $n = 50$	$\eta = .25$	123	69	82	76
	$\eta = .1$	117	75	90	76
	$\eta = .001$	99	85	117	95
Pen3 Start 3 $n = 100$	$\eta = .25$	97	87	80	83
	$\eta = .1$	108	98	109	82
	$\eta = .001$	112	97	132	96
PSP Start 1 $n = 50$	$\eta = .25$	5	5	5	5
	$\eta = .1$	7	7	7	7
	$\eta = .001$	7	7	7	7

KEY

CG - traditional conjugate-gradient method.
PCG - algorithm CG with diagonal preconditioning.
BCG - Beale's method with Powell restarts.
PBCG - diagonally preconditioned Beale's method.

TABLE A2
Number of *Function Evaluations* Required By
Limited-memory Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM	η	Shanno	PLM1	PLM2	PLMA
Pen1 Start 3 $n = 50$	$\eta = .25$	27	53	53	53
	$\eta = .1$	37	28	29	27
	$\eta = .001$	49	34	33	32
Pen1 Start 3 $n = 100$	$\eta = .25$	28	46	44	40
	$\eta = .1$	13	10	10	10
	$\eta = .001$	14	10	10	10
Pen1 Start 2 $n = 50$	$\eta = .25$	21	21	20	21
	$\eta = .1$	23	28	28	32
	$\eta = .001$	25	23	28	29
Pen1 Start 2 $n = 100$	$\eta = .25$	19	23	20	20
	$\eta = .1$	45	41	41	44
	$\eta = .001$	39	48	48	57
Pen2 Start 5 $n = 50$	$\eta = .25$	15	35	51	67
	$\eta = .1$	33	36	33	47
	$\eta = .001$	42	67	77	112
Pen2 Start 5 $n = 100$	$\eta = .25$	8	10	15	28
	$\eta = .1$	14	12	13	13
	$\eta = .001$	21	18	18	23

KEY

Shanno - Shanno's method.
PLM1 - preconditioned one-step BFGS.
PLM2 - preconditioned two-step BFGS.
PLMA - method PLM2 with accumulated step.

TABLE A2 (continued)
Number of *Function Evaluations* Required by
Limited-memory Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM	η	Shanno	PLM1	PLM2	PLMA
Pen2 Start 3 $n = 50$	$\eta = .25$	74	58	70	118
	$\eta = .1$	99	80	68	76
	$\eta = .001$	127	106	82	71
Pen2 Start 3 $n = 100$	$\eta = .25$	17	12	16	28
	$\eta = .1$	16	17	15	18
	$\eta = .001$	32	30	23	29
Pen3 Start 1 $n = 50$	$\eta = .25$	87	64	85	65
	$\eta = .1$	81	82	90	62
	$\eta = .001$	80	92	104	71
Pen3 Start 1 $n = 100$	$\eta = .25$	89	115	95	77
	$\eta = .1$	84	120	109	87
	$\eta = .001$	81	132	94	79
Pen3 Start 3 $n = 50$	$\eta = .25$	83	89	101	76
	$\eta = .1$	75	89	107	76
	$\eta = .001$	85	117	116	71
Pen3 Start 3 $n = 100$	$\eta = .25$	85	82	87	85
	$\eta = .1$	79	84	111	94
	$\eta = .001$	94	132	124	83
PSP Start 1 $n = 50$	$\eta = .25$	4	4	4	4
	$\eta = .1$	6	7	6	6
	$\eta = .001$	7	7	7	7

KEY

Shanno - Shanno's method.
PLM1 - preconditioned one-step BFGS.
PLM2 - preconditioned two-step BFGS.
PLMA - method PLM2 with accumulated step.

TABLE A3
Number of *Function Evaluations* Required
by Conjugate-gradient Type Methods
on General Problems

PROBLEM	η	CG	BCG	PCG	PBCG
Chebyquad Start 2 $n = 6$	$\eta = .25$	23	14	22	18
	$\eta = .1$	25	17	20	18
	$\eta = .001$	30	22	29	26
Chebyquad Start 2 $n = 8$	$\eta = .25$	31	22	24	26
	$\eta = .1$	31	30	26	27
	$\eta = .001$	32	34	34	37
Chebyquad Start 2 $n = 20$	$\eta = .25$	109	85	89	79
	$\eta = .1$	98	119	85	81
	$\eta = .001$	138	132	94	86
Watson Start 1 $n = 6$	$\eta = .25$	556	135	>700	546
	$\eta = .1$	456	81	>700	200
	$\eta = .001$	470	161	>700	609
GenRose Start 2 $n = 50$	$\eta = .25$	652	551	216	190
	$\eta = .1$	667	599	219	202
	$\eta = .001$	748	664	269	250
GenRose Start 2 $n = 100$	$\eta = .25$	1197	1057	325	318
	$\eta = .1$	1247	1070	357	338
	$\eta = .001$	1354	1239	494	457
Calvar1 Start 1 $n = 50$	$\eta = .25$	>2000	>2000	346	427
	$\eta = .1$	>2000	>2000	354	425
	$\eta = .001$	>2000	>2000	419	433
Calvar1 Start 1 $n = 100$	$\eta = .25$	>2000	>2000	815	828
	$\eta = .1$	>2000	>2000	821	857
	$\eta = .001$	>2000	>2000	929	985

KEY

CG - traditional conjugate-gradient method.
PCG - algorithm CG with diagonal preconditioning.
BCG - Beale's method with Powell restarts.
PBCG - diagonally preconditioned Beale's method.

TABLE A3 (continued)
Number of *Function Evaluations* Required
by Conjugate-gradient Type Methods
on General Problems

PROBLEM	η	CG	BCG	PCG	PBCG
Calvar2 Start 1 $n = 50$	$\eta = .25$	239	178	159	126
	$\eta = .1$	239	191	164	129
	$\eta = .001$	251	190	168	124
Calvar2 Start 1 $n = 100$	$\eta = .25$	520	409	299	284
	$\eta = .1$	397	349	305	227
	$\eta = .001$	475	371	312	238
Calvar3 Start 1 $n = 50$	$\eta = .25$	971	709	162	157
	$\eta = .1$	775	709	159	145
	$\eta = .001$	850	784	172	160
Calvar3 Start 1 $n = 100$	$\eta = .25$	1903	1363	292	277
	$\eta = .1$	1927	1340	275	280
	$\eta = .001$	1695	1507	290	284
Var(0) Start 4 $n = 100$	$\eta = .25$	200	200	675	673
	$\eta = .1$	200	200	701	529
	$\eta = .001$	200	200	774	512
Var(1) Start 4 $n = 50$	$\eta = .25$	150	130	322	315
	$\eta = .1$	150	130	334	272
	$\eta = .001$	151	131	355	240
Var(1) Start 4 $n = 100$	$\eta = .25$	344	284	638	639
	$\eta = .1$	344	284	665	555
	$\eta = .001$	347	289	717	544

KEY

CG - traditional conjugate-gradient method.
PCG - algorithm CG with diagonal preconditioning.
BCG - Beale's method with Powell restarts.
PBCG - diagonally preconditioned Beale's method.

TABLE A3 (continued)
Number of *Function Evaluations* Required
by Conjugate-gradient Type Methods
on General Problems

PROBLEM			η	CG	BCG	PCG	PBCG
QOR	Start 1	$n = 50$	$\eta = .25$	27	27	29	29
			$\eta = .1$	27	27	29	29
			$\eta = .001$	27	27	29	29
GOR	Start 1	$n = 50$	$\eta = .25$	87	81	72	71
			$\eta = .1$	87	81	76	76
			$\eta = .001$	108	99	95	92
ChnRose	Start 6	$n = 25$	$\eta = .25$	81	82	74	57
			$\eta = .1$	84	78	82	63
			$\eta = .001$	100	106	86	95

KEY

CG - traditional conjugate-gradient method.
PCG - algorithm CG with diagonal preconditioning.
BCG - Beale's method with Powell restarts.
PBCG - diagonally preconditioned Beale's method.

TABLE A4
Number of *Function Evaluations* Required by
Limited-memory Quasi-Newton Methods
on General Problems

PROBLEM	η	Shanno	PLM1	PLM2	PLMA
Chebyquad Start 2 $n = 6$	$\eta = .25$	19	18	17	17
	$\eta = .1$	20	22	17	18
	$\eta = .001$	24	27	23	26
Chebyquad Start 2 $n = 8$	$\eta = .25$	27	26	28	21
	$\eta = .1$	29	27	30	21
	$\eta = .001$	43	34	34	26
Chebyquad Start 2 $n = 20$	$\eta = .25$	72	80	80	75
	$\eta = .1$	71	95	86	71
	$\eta = .001$	88	87	83	90
Watson Start 1 $n = 6$	$\eta = .25$	174	>700	>700	294
	$\eta = .1$	175	>700	>700	406
	$\eta = .001$	186	>700	>700	316
GenRose Start 2 $n = 50$	$\eta = .25$	243	199	203	201
	$\eta = .1$	250	210	207	263
	$\eta = .001$	285	261	271	330
GenRose Start 2 $n = 100$	$\eta = .25$	403	330	328	365
	$\eta = .1$	416	348	361	410
	$\eta = .001$	526	480	478	528
Calvar1 Start 1 $n = 50$	$\eta = .25$	>2000	451	428	366
	$\eta = .1$	>2000	464	406	401
	$\eta = .001$	>2000	478	457	456
Calvar1 Start 1 $n = 100$	$\eta = .25$	>2000	841	879	819
	$\eta = .1$	>2000	951	904	854
	$\eta = .001$	>2000	992	1025	905

KEY

Shanno - Shanno's method.
PLM1 - preconditioned one-step BFGS.
PLM2 - preconditioned two-step BFGS.
PLMA - method PLM2 with accumulated step.

TABLE A4 (continued)
Number of *Function Evaluations* Required by
Limited-memory Quasi-Newton Methods
on General Problems

PROBLEM	η	Shanno	PLM1	PLM2	PLMA
Calvar2 Start 1 $n = 50$	$\eta = .25$	382	168	127	106
	$\eta = .1$	275	159	134	118
	$\eta = .001$	192	168	158	123
Calvar2 Start 1 $n = 100$	$\eta = .25$	690	253	351	204
	$\eta = .1$	548	308	310	206
	$\eta = .001$	381	328	330	228
Calvar3 Start 1 $n = 50$	$\eta = .25$	1040	162	165	152
	$\eta = .1$	861	172	170	155
	$\eta = .001$	611	186	178	161
Calvar3 Start 1 $n = 100$	$\eta = .25$	1891	308	308	270
	$\eta = .1$	>2000	304	306	281
	$\eta = .001$	1322	318	314	284
Var(0) Start 4 $n = 100$	$\eta = .25$	463	614	603	475
	$\eta = .1$	413	762	746	494
	$\eta = .001$	201	787	783	579
Var(1) Start 4 $n = 50$	$\eta = .25$	184	276	321	199
	$\eta = .1$	139	358	303	224
	$\eta = .001$	128	347	361	261
Var(1) Start 4 $n = 100$	$\eta = .25$	398	664	545	497
	$\eta = .1$	412	690	654	534
	$\eta = .001$	288	720	713	547

KEY

Shanno - Shanno's method.
PLM1 - preconditioned one-step BFGS.
PLM2 - preconditioned two-step BFGS.
PLMA - method PLM2 with accumulated step.

TABLE A4 (continued)
Number of *Function Evaluations* Required by
Limited-memory Quasi-Newton Methods
on General Problems

PROBLEM			η	Shanno	PLM1	PLM2	PLMA
QOR	Start 1	$n = 50$	$\eta = .25$	28	29	29	29
			$\eta = .1$	27	29	27	29
			$\eta = .001$	27	29	27	29
GOR	Start 1	$n = 50$	$\eta = .25$	73	73	79	71
			$\eta = .1$	78	81	77	76
			$\eta = .001$	94	95	94	97
ChnRose	Start 6	$n = 25$	$\eta = .25$	60	69	57	82
			$\eta = .1$	82	102	103	76
			$\eta = .001$	95	108	116	119

KEY

Shanno - Shanno's method.
PLM1 - preconditioned one-step BFGS.
PLM2 - preconditioned two-step BFGS.
PLMA - method PLM2 with accumulated step.

TABLE A5
Number of *Function Evaluations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM	η	PLMA	MNM	QNM
Pen1 Start 3 $n = 50$	$\eta = .25$	53	18	33
	$\eta = .1$	27	25	26
	$\eta = .001$	32	29	31
Pen1 Start 3 $n = 100$	$\eta = .25$	40	NR	NR
	$\eta = .1$	10	NR	NR
	$\eta = .001$	10	NR	NR
Pen1 Start 2 $n = 50$	$\eta = .25$	21	11	22
	$\eta = .1$	32	16	30
	$\eta = .001$	29	18	22
Pen1 Start 2 $n = 100$	$\eta = .25$	20	NR	NR
	$\eta = .1$	44	NR	NR
	$\eta = .001$	57	NR	NR
Pen2 Start 5 $n = 50$	$\eta = .25$	67	15	214
	$\eta = .1$	47	18	239
	$\eta = .001$	112	19	290
Pen2 Start 5 $n = 100$	$\eta = .25$	28	NR	NR
	$\eta = .1$	13	NR	NR
	$\eta = .001$	23	NR	NR
Pen2 Start 5 $n = 100$	$\eta = .25$	28	NR	NR
	$\eta = .1$	13	NR	NR
	$\eta = .001$	23	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A5 (continued)
Number of *Function Evaluations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM			η	PLMA	MNM	QNM
Pen2	Start 3	$n = 50$	$\eta = .25$	118	17	242
			$\eta = .1$	76	31	322
			$\eta = .001$	71	26	341
Pen2	Start 3	$n = 100$	$\eta = .25$	28	NR	NR
			$\eta = .1$	18	NR	NR
			$\eta = .001$	29	NR	NR
Pen3	Start 1	$n = 50$	$\eta = .25$	65	14	115
			$\eta = .1$	62	20	113
			$\eta = .001$	71	24	146
Pen3	Start 1	$n = 100$	$\eta = .25$	77	NR	NR
			$\eta = .1$	87	NR	NR
			$\eta = .001$	79	NR	NR
Pen3	Start 3	$n = 50$	$\eta = .25$	76	44	135
			$\eta = .1$	76	44	150
			$\eta = .001$	71	48	155
Pen3	Start 3	$n = 100$	$\eta = .25$	85	NR	NR
			$\eta = .1$	94	NR	NR
			$\eta = .001$	83	NR	NR
PSP	Start 1	$n = 50$	$\eta = .25$	4	2	5
			$\eta = .1$	6	2	7
			$\eta = .001$	7	2	7

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A6
Number of *Function Evaluations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
Chebyquad	Start 2	$n = 6$	$\eta = .25$	17	18	13
			$\eta = .1$	18	39	15
			$\eta = .001$	26	56	19
Chebyquad	Start 2	$n = 8$	$\eta = .25$	21	38	21
			$\eta = .1$	21	64	25
			$\eta = .001$	26	68	36
Chebyquad	Start 2	$n = 20$	$\eta = .25$	75	121	65
			$\eta = .1$	71	116	67
			$\eta = .001$	90	161	92
Watson	Start 1	$n = 6$	$\eta = .25$	294	11	28
			$\eta = .1$	406	15	37
			$\eta = .001$	316	27	55
GenRose	Start 2	$n = 50$	$\eta = .25$	201	202	287
			$\eta = .1$	263	257	323
			$\eta = .001$	330	392	412
GenRose	Start 2	$n = 100$	$\eta = .25$	365	NR	NR
			$\eta = .1$	410	NR	NR
			$\eta = .001$	528	NR	NR
Calvar1	Start 1	$n = 50$	$\eta = .25$	366	9	191
			$\eta = .1$	401	11	214
			$\eta = .001$	456	17	269
Calvar1	Start 1	$n = 100$	$\eta = .25$	819	NR	NR
			$\eta = .1$	854	NR	NR
			$\eta = .001$	905	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A6 (continued)
Number of *Function Evaluations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
Calvar2	Start 1	$n = 50$	$\eta = .25$	106	4	52
			$\eta = .1$	118	4	54
			$\eta = .001$	123	6	67
Calvar2	Start 1	$n = 100$	$\eta = .25$	204	NR	NR
			$\eta = .1$	206	NR	NR
			$\eta = .001$	228	NR	NR
Calvar3	Start 1	$n = 50$	$\eta = .25$	152	6	114
			$\eta = .1$	155	7	135
			$\eta = .001$	161	11	161
Calvar3	Start 1	$n = 100$	$\eta = .25$	270	NR	NR
			$\eta = .1$	281	NR	NR
			$\eta = .001$	284	NR	NR
Var(0)	Start 4	$n = 100$	$\eta = .25$	475	NR	NR
			$\eta = .1$	494	NR	NR
			$\eta = .001$	579	NR	NR
Var(1)	Start 4	$n = 50$	$\eta = .25$	199	3	102
			$\eta = .1$	224	3	101
			$\eta = .001$	261	4	102
Var(1)	Start 4	$n = 100$	$\eta = .25$	497	NR	NR
			$\eta = .1$	534	NR	NR
			$\eta = .001$	547	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A6 (continued)
Number of *Function Evaluations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
QOR	Start 1	$n = 50$	$\eta = .25$	29	3	39
			$\eta = .1$	29	3	27
			$\eta = .001$	29	3	27
GOR	Start 1	$n = 50$	$\eta = .25$	71	5	59
			$\eta = .1$	76	5	59
			$\eta = .001$	97	7	72
ChnRose	Start 6	$n = 25$	$\eta = .25$	82	28	97
			$\eta = .1$	76	48	122
			$\eta = .001$	119	47	164

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A7
Number of *Iterations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM			η	PLMA	MNM	QNM
Pen1	Start 3	$n = 50$	$\eta = .25$	22	17	27
			$\eta = .1$	8	9	8
			$\eta = .001$	8	7	8
Pen1	Start 3	$n = 100$	$\eta = .25$	17	NR	NR
			$\eta = .1$	2	NR	NR
			$\eta = .001$	2	NR	NR
Pen1	Start 2	$n = 50$	$\eta = .25$	8	11	17
			$\eta = .1$	11	6	10
			$\eta = .001$	7	5	5
Pen1	Start 2	$n = 100$	$\eta = .25$	9	NR	NR
			$\eta = .1$	13	NR	NR
			$\eta = .001$	14	NR	NR
Pen2	Start 5	$n = 50$	$\eta = .25$	31	14	98
			$\eta = .1$	15	6	72
			$\eta = .001$	27	5	62
Pen2	Start 5	$n = 100$	$\eta = .25$	13	NR	NR
			$\eta = .1$	6	NR	NR
			$\eta = .001$	6	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - *quasi-Newton method.*
NR - Not run.

TABLE A7 (continued)
Number of *Iterations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on Problems With Clustered Eigenvalues

PROBLEM			η	PLMA	MNM	QNM
Pen2	Start 3	$n = 50$	$\eta = .25$	52	17	134
			$\eta = .1$	28	9	99
			$\eta = .001$	15	6	73
Pen2	Start 3	$n = 100$	$\eta = .25$	14	NR	NR
			$\eta = .1$	7	NR	NR
			$\eta = .001$	7	NR	NR
Pen3	Start 1	$n = 50$	$\eta = .25$	35	10	57
			$\eta = .1$	30	8	51
			$\eta = .001$	28	8	54
Pen3	Start 1	$n = 100$	$\eta = .25$	39	NR	NR
			$\eta = .1$	44	NR	NR
			$\eta = .001$	34	NR	NR
Pen3	Start 3	$n = 50$	$\eta = .25$	40	40	67
			$\eta = .1$	38	12	63
			$\eta = .001$	28	11	56
Pen3	Start 3	$n = 100$	$\eta = .25$	49	NR	NR
			$\eta = .1$	48	NR	NR
			$\eta = .001$	35	NR	NR
PSP	Start 1	$n = 50$	$\eta = .25$	3	2	4
			$\eta = .1$	3	2	3
			$\eta = .001$	3	2	3

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A8
Number of *Iterations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
Chebyquad	Start 2	$n = 6$	$\eta = .25$	8	4	8
			$\eta = .1$	8	4	8
			$\eta = .001$	9	6	7
Chebyquad	Start 2	$n = 8$	$\eta = .25$	10	6	14
			$\eta = .1$	10	8	12
			$\eta = .001$	10	10	14
Chebyquad	Start 2	$n = 20$	$\eta = .25$	38	29	32
			$\eta = .1$	33	24	28
			$\eta = .001$	33	30	28
Watson	Start 1	$n = 6$	$\eta = .25$	150	11	25
			$\eta = .1$	188	7	18
			$\eta = .001$	129	8	19
GenRose	Start 2	$n = 50$	$\eta = .25$	108	62	128
			$\eta = .1$	119	66	118
			$\eta = .001$	119	88	118
GenRose	Start 2	$n = 100$	$\eta = .25$	191	NR	NR
			$\eta = .1$	192	NR	NR
			$\eta = .001$	188	NR	NR
Calvar1	Start 1	$n = 50$	$\eta = .25$	194	7	162
			$\eta = .1$	204	6	89
			$\eta = .001$	205	6	88
Calvar1	Start 1	$n = 100$	$\eta = .25$	423	NR	NR
			$\eta = .1$	429	NR	NR
			$\eta = .001$	416	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A8 (continued)
Number of *Iterations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
Calvar2	Start 1	$n = 50$	$\eta = .25$	64	4	28
			$\eta = .1$	61	4	28
			$\eta = .001$	60	4	28
Calvar2	Start 1	$n = 100$	$\eta = .25$	112	NR	NR
			$\eta = .1$	107	NR	NR
			$\eta = .001$	113	NR	NR
Calvar3	Start 1	$n = 50$	$\eta = .25$	80	6	90
			$\eta = .1$	78	5	59
			$\eta = .001$	77	5	59
Calvar3	Start 1	$n = 100$	$\eta = .25$	143	NR	NR
			$\eta = .1$	142	NR	NR
			$\eta = .001$	138	NR	NR
Var(0)	Start 4	$n = 100$	$\eta = .25$	266	NR	NR
			$\eta = .1$	255	NR	NR
			$\eta = .001$	289	NR	NR
Var(1)	Start 4	$n = 50$	$\eta = .25$	115	3	52
			$\eta = .1$	117	3	51
			$\eta = .001$	129	3	51
Var(1)	Start 4	$n = 100$	$\eta = .25$	273	NR	NR
			$\eta = .1$	274	NR	NR
			$\eta = .001$	271	NR	NR

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

TABLE A8 (continued)
Number of *Iterations* Required
by Preconditioned Limited-Memory
Methods vs Modified-Newton and
Full Quasi-Newton Methods
on General Problems

PROBLEM			η	PLMA	MNM	QNM
QOR	Start 1	$n = 50$	$\eta = .25$	14	3	23
			$\eta = .1$	14	3	13
			$\eta = .001$	14	3	13
GOR	Start 1	$n = 50$	$\eta = .25$	41	5	29
			$\eta = .1$	41	5	29
			$\eta = .001$	42	5	29
ChnRose	Start 6	$n = 25$	$\eta = .25$	40	15	48
			$\eta = .1$	37	16	46
			$\eta = .001$	43	12	47

KEY

PLMA - preconditioned two-step BFGS with accumulated step.
MNM - modified Newton method.
QNM - quasi-Newton method.
NR - Not run.

References

Beale, E. M. L. (1972). A derivation of conjugate gradients, in: *Numerical Methods for Nonlinear Optimization* (F. A. Lootsma, ed.), pp. 39–43, Academic Press, London and New York.

Brent, R. P. (1973). *Algorithms for Minimization without Derivatives*, Prentice-Hall, New Jersey.

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms, *J. Inst. Maths Applics.*, **6**, pp. 76–90.

Concus P., Golub, G. H. and O'Leary, P. (1976). A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in: *Sparse Matrix Computations* (J. R. Bunch and D. J. Rose, eds.), pp. 309–332, Academic Press, London and New York.

Curtis, A. R., Powell, M. J. D. and Reid, J. K. (1974). On the estimation of sparse Jacobian matrices, *J. Inst. Maths Applics.*, **13**, pp. 117–119.

Davidon, W. (1959). Variable metric methods for minimization, A.E.C. Res. and Develop. Report ANL-5990, Argonne National Laboratory.

Dennis, J. E. and Moré J. J. (1977). Quasi-Newton methods, motivation and theory, *SIAM Review*, **19**, pp. 46–89.

Fletcher R. (1965). Function minimization without evaluating derivatives - a review, *Comput. J.*, **8**, pp. 33–41.

Fletcher. R. (1972). Conjugate direction methods, in: *Numerical Methods for Unconstrained Optimization* (W. Murray, ed.), pp. 73–86, Academic Press, London and New York.

Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients, *Comput. J.*, **7**, pp. 149–154.

Gill, P. E. and Murray W. (1972a). Quasi-Newton methods for unconstrained optimization, *J. Inst. Maths Applics.*, **9**, pp. 91–108.

Gill, P. E., Murray W. and Pitfield, R. A. (1972b). The implementation of two revised Quasi-Newton algorithms for unconstrained optimization. Report NAC 11, National Physical Laboratory, England.

Gill, P. E. and Murray W. (1973). The numerical solution of a problem in the calculus of variations, in: *Recent Mathematical Developments in Control*

(D. J. Bell, ed.), pp. 97–122, Academic Press, London and New York.

Gill, P. E. and Murray W. (1974a). Newton-type methods for unconstrained and linearly constrained optimization, *Math. Prog.* **28**, pp. 311–350.

Gill, P. E. and Murray W. (eds.) (1974b). *Numerical Methods for Constrained Optimization*, Academic Press, London and New York.

Gill, P. E. and Murray W. (1974c). Safeguarded steplength algorithms for optimization using descent methods, Report NAC 37, National Physical Laboratory, England.

Gill, P. E. and Murray W. (1976). Minimization subject to bounds on the variables. Report NAC 71, National Physical Laboratory, England.

Gill, P. E. and Murray W. (1979). Performance evaluation for nonlinear optimization, in: *Performance Evaluation for Numerical Software* (L. Fosdick, ed.), North-Holland.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards*, **49**, pp. 409–436.

McCormick, G. P. and Pearson, J. D. (1969). Variable metric methods and unconstrained optimization, in: *Optimization* (R. Fletcher, ed.), pp. 307–325, Academic Press, London and New York.

Meijerink, J. A. and Van der Vorst, H. A. (1977). An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comp.*, **31**, pp. 148–162.

Munksgaard, N. (1979). Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients, Report CSS 67, A. E. R. E. Harwell, England.

Murray, W. and Wright M. H. (1978). Projected Lagrangian methods based on the trajectories of penalty and barrier functions, Report SOL 78–23, Department of Operations Research, Stanford University.

Nazareth, L. (1979). A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms, *SIAM J. Num. Anal.* **16**, pp. 794–800.

Nazareth, L. and Nocedal, J. (1978). A study of conjugate gradient methods, Report SOL 78-29, Operations Research Department, Stanford University.

Oren, S. S. (1974). Self-scaling variable metric (SSVM) algorithms II: Implementation and experiments, *Management Science*, **20**, pp. 863–874.

Oren, S. S. and Spedicato, E. (1976). Optimal conditioning of self-scaling and variable metric algorithms, *Math. Prog.*, **10**, pp. 70–90.

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

Perry, A. (1977). A class of conjugate gradient algorithms with a two-step variable-metric memory, Discussion paper 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University.

Polak, E. (1971). *Computational Methods in Optimization: a Unified Approach*, Academic Press, London and New York.

Powell, M. J. D. (1976). Some convergence properties of the conjugate gradient method, *Math. Prog.* **11**, pp. 42–49.

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method, *Math. Prog.* **12**, pp. 241–254.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function, *Comput. J.* **3**, pp. 175–184.

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

Shanno, D. F. (1978b). On variable metric methods for sparse Hessians, MIS Report 26, Department of Management Information Systems, University of Arizona.

Toint, Ph. L. (1977). On sparse and symmetric matrix updating subject to a linear equation, *Math. Comp.* **31**, pp. 954–961.

Toint, Ph. L. (1978). Some numerical results using a sparse matrix updating formula in unconstrained optimization, *Math. Comp.* **32**, pp. 839–851.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 79-15	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CONJUGATE-GRADIENT METHODS FOR LARGE-SCALE NONLINEAR OPTIMIZATION		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL REPROT 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip E. Gill and Walter Murray		8. CONTRACT OR GRANT NUMBER(s) DAAG29-79-C-0110
9. PERFORMING ORGANIZATION NAME AND ADDRESS Operations Research Department - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Office Box CM, Duke Station Durham, NC 27706		12. REPORT DATE October 1979 13. NUMBER OF PAGES 61
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public relase and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Large-scale minimization
Conjugate-gradient methods
Limited-memory quasi-Newton methods
Computer implementation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SOL 79-15 Philip E. Gill and Walter Murray

CONJUGATE-GRADIENT METHODS
FOR LARGE-SCALE NONLINEAR OPTIMIZATION

In this paper we discuss several recent conjugate-gradient type methods for solving large-scale nonlinear optimization problems. We demonstrate how the performance of these methods can be significantly improved by careful implementation. A method based upon iterative preconditioning will be suggested which performs reasonably efficiently on a wide variety of significant test problems.

Our results indicate that nonlinear conjugate-gradient methods behave in a similar way to conjugate-gradient methods for the solution of systems of linear equations. These methods work best on problems whose Hessian matrices have sets of clustered eigenvalues. On more general problems, however, even the best method may require a prohibitively large number of iterations. We present numerical evidence that indicates that the use of theoretical analysis to predict the performance of algorithms on general problems is not straightforward.