Mathematical Programming 27 (1983) 155–175
North-Holland

QN-LIKE VARIABLE STORAGE CONJUGATE GRADIENTS*

A. BUCKLEY and A. LENIR
Mathematics Department, Concordia University, Montreal, Quebec, Canada

Received 25 February 1982
Revised manuscript received 1 November 1982

Both conjugate gradient and quasi-Newton methods are quite successful at minimizing smooth nonlinear functions of several variables, and each has its advantages. In particular, conjugate gradient methods require much less storage to implement than a quasi-Newton code and therefore find application when storage limitations occur. They are, however, slower, so there have recently been attempts to combine CG and QN algorithms so as to obtain an algorithm with good convergence properties and low storage requirements. One such method is the code CONMIN due to Shanno and Phua; it has proven quite successful but it has one limitation. It has no middle ground, in that it either operates as a quasi-Newton code using $O(n^2)$ storage locations, or as a conjugate gradient code using $7n$ locations, but it cannot take advantage of the not unusual situation where more than $7n$ locations are available, but a quasi-Newton code requires an excessive amount of storage.

In this paper we present a way of looking at conjugate gradient algorithms which was in fact given by Shanno and Phua but which we carry further, emphasize and clarify. This applies in particular to Beale's 3-term recurrence relation. Using this point of view, we develop a new combined CG–QN algorithm which can use whatever storage is available; CONMIN occurs as a special case. We present numerical results to demonstrate that the new algorithm is never worse than CONMIN and that it is almost always better if even a small amount of extra storage is provided.

Key words: Minimization, Conjugate Gradient, Quasi-Newton, Variable Storage, Reduced Storage.

0. Introduction

The purpose of this paper is to present a new conjugate gradient algorithm for minimization of a smooth function $f(x)$, where $x = (x_1, \ldots, x_n)^T$. A main objective of the algorithm is to allow the use of a variable amount of storage, according to availability, in such a way that more storage will mean improved performance. It is intended to publish an implementation of the algorithm separately. A second purpose of this paper is to emphasize a point of view of conjugate gradient algorithms that has been made in [3].

To motivate the stated objective, recall that conjugate gradient algorithms have one main advantage: they require only some small multiple of n locations of storage for their implementation. Currently, there are successful conjugate gradient

* The authors wish to express their appreciation for the support of the Natural Sciences and Engineering Research Council through Operating Grant A8962 (Buckley) and of the National Research Council of Canada through a Postgraduate Scholarship (LeNir).

algorithms readily available from software sources, but there is one feature that these generally lack, namely, the ability to use available space. To see why this is relevant, consider a problem which is large enough that one is not able to provide the $O(n^2)$ locations needed to use a quasi-Newton algorithm. Take, for example, $n = 500$ on a medium size machine. In this case, one might still expect to be able to provide working storage in the order of some moderate multiple of n, say $10n$ or $20n$. The current algorithms, at least those available as software, each use a fixed amount of storage, typically $4n$ or $7n$, and are not able to make full use of the space actually available.

There have been earlier attempts to write variable storage codes, see e.g. [1], [7] and [8] and some numerical results based on varying the storage were quite encouraging [8]. Here we will describe an algorithm which can effectively use whatever space is available. It is related to one given by Shanno [12], but our derivation will be a bit different and is intended to emphasize a particular point of view of conjugate gradients, as already mentioned. Both our algorithm and Shanno's also have the feature that they are not dependent on the use of highly accurate line searches. Numerical results will be presented to demonstrate that the use of additional space can aid performance and indeed we think we successfully substantiate our claim that our algorithm does indeed meet its objective.

1. Preliminaries

We let a small letter, e.g. v, denote a column vector, so v^T is always a row; these are also used for indices. Greek letters denote a scalar; capitals are used for matrices. We use 'H' generically for a quasi-Newton update matrix. The algorithm proceeds from a given point x_0 by constructing directions $d_1, d_2, \ldots$ and setting $x_i = x_{i-1} + \alpha_i d_i$ for suitable α_i. We will write $s_i = x_i - x_{i-1}$ and, with $g_i \equiv g(x_i) \equiv \nabla f(x_i)$, also $y_i = g_i - g_{i-1}$.

It is now firmly established (see e.g. [2, 7]) that conjugate gradient (CG) directions have close ties with BFGS quasi-Newton updates. Here the connection will be of fundamental importance, so we will introduce the BFGS update forthwith; thus, given H, we write[1]

$$H^* = U(H, i) \tag{1}$$

where

$$U(H, i) \equiv H - \frac{s_i y_i^T H + H y_i s_i^T}{s_i^T y_i} + \left(1 + \frac{y_i^T H y_i}{s_i^T y_i}\right) \frac{s_i s_i^T}{s_i^T y_i}. \tag{2}$$

Familiarity with quasi-Newton algorithms will be assumed.

For a quadratic function, say $f(x) = \frac{1}{2} x^T A x + b^T x$, we recall that $A s_i = y_i$. Also, the directions $d_1, d_2, \ldots$ are conjugate (with respect to A) if $d_i^T A d_j = 0$ for $i \neq j$.

[1] This is most easily read as 'H^* is Update of H at ith step'.
H^* U (H, i)

Since $s_i = \alpha_i d_i$, the conjugacy relation can be written as $y_i^T d_j = 0$ for $i \neq j$. When we write subsequently of the 'quadratic case', it will be naturally assumed that line searches are exact.

2. Preconditioning, Beale restarts and updating

In this section we will demonstrate how the preconditioned CG algorithm can be written in a quasi-Newton like manner, and we will explain how this gives a particular way of viewing CG algorithms. We will also show that the Beale restart algorithm is actually a preconditioned CG algorithm, so that it can be viewed similarly.

The CG algorithm is now commonly written with a preconditioner H. It is assumed that H is positive definite and then, given x_0, one computes $d_1 = -Hg_0$ and iterates with

$$x_i = x_{i-1} + \alpha_i d_i, \tag{3a}$$

$$\beta_i = \frac{g_i^T H y_i}{d_i^T y_i}, \tag{3b}$$

$$d_{i+1} = -Hg_i + \beta_i d_i. \tag{3c}$$

When f is quadratic, it is well known (especially when $H = I$) that β_i may be written alternately as

$$\beta_i = \frac{g_i^T H g_i}{g_{i-1}^T H g_{i-1}} \quad \text{or} \quad \beta_i = \frac{g_i^T H y_i}{g_{i-1}^T H g_{i-1}}.$$

For our purposes, the given form (3b) is the most useful, as will become clear in what follows. We choose it as well because it ensures, with no assumption of an exact line search or quadratic behavior, that $y_i^T d_{i+1} = 0$.

When $H = I$, Perry [10] observed that (3c) can be written in an alternate form. Not surprisingly, one may do the same for any H. Thus we recall that $\alpha_i d_i = s_i$, substitute (3b), and observe that (3c) can be written as

$$d_{i+1} = -\left(H - \frac{s_i y_i^T H}{s_i^T y_i}\right) g_i \equiv -Q_i g_i. \tag{4}$$

Equation (4) emphasizes that a conjugate gradient search direction can be computed from an equation which is of the same form as that used to compute a quasi-Newton direction. It also demonstrates, in part, why CG algorithms have difficulty with the line search subproblem: the matrix Q_i is not positive definite and it is in fact singular.

Before examining Q_i in more detail, let us consider Beale's recurrence relation for generating conjugate directions. As before, we have $x_i = x_{i-1} + \alpha_i d_i$, but d_1 is now arbitrary (actually, d_1 must be downhill, whereas for the ordinary conjugate gradient algorithm it is required that $d_1 = -Hg_0$). For what will follow, we change

the indexing and let d_r be the given direction[2]; at the start $d_r \equiv d_1$ and $r = 1$. The Beale recurrence can then be written as

$$x_i = x_{i-1} + \alpha_i d_i, \qquad i \geq r, \tag{5a}$$

$$\beta_i = \frac{g_i^{\mathrm{T}} \dot{H} y_i}{d_i^{\mathrm{T}} y_i}, \qquad i \geq r, \tag{5b}$$

$$\kappa_r = 0, \kappa_i = \frac{g_i^{\mathrm{T}} \dot{H} y_r}{d_r^{\mathrm{T}} y_r}, \qquad i > r, \tag{5c}$$

$$d_{i+1} = -\dot{H} g_i + \beta_i d_i + \kappa_i d_r, \quad i \geq r. \tag{5d}$$

Observe that Beale's recurrence is normally written without the positive definite preconditioner $\dot{H}$, but its inclusion causes no complication and we will refer to it later.

There are two ways that (5d) may be rewritten using (5b) and (5c). First, one may write

$$d_{r+1} = -\left(\dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H}}{s_r^{\mathrm{T}} y_r}\right) g_r \qquad \equiv -\dot{Q}_r g_r$$

$$d_{i+1} = -\left(\dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H}}{s_r^{\mathrm{T}} y_r} - \frac{s_i y_i^{\mathrm{T}} \dot{H}}{s_i^{\mathrm{T}} y_i}\right) g_i \equiv -\dot{Q}_i g_i, \quad i > r.$$

These equations once again serve to indicate the quasi-Newton like form of Beale's equations, but for our purposes they are not the most useful. Instead we write

$$d_{r+1} = -\left(\dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H}}{s_r^{\mathrm{T}} y_r}\right) g_r \qquad \equiv -H_r g_r, \tag{6a}$$

$$d_{i+1} = -\left(\dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H}}{s_r^{\mathrm{T}} y_r}\right) g_i + \beta_i d_i = -H_r g_i + \beta_i d_i, \quad i > r. \tag{6b}$$

Upon first examination, comparing (6b) to (3c), the formulae (6) appear to define a sequence of preconditioned conjugate gradient steps with preconditioner H_r. However, for that to be true, the formula for β_i in (6b) should be

$$\beta_i = \frac{g_i^{\mathrm{T}} H_r y_i}{d_i^{\mathrm{T}} y_i}, \tag{7}$$

whereas it is in fact given, according to (5b), as

$$\beta_i = \frac{g_i^{\mathrm{T}} \dot{H} y_i}{d_i^{\mathrm{T}} y_i}. \tag{8}$$

As has been the case in the past, we will consider two forms of a conjugate gradient algorithm to be equivalent if they generate the same points when applied with

[2] The restart direction has commonly been written as d_1, but here we have chosen to write it as d_r, so that one may read d_r as d_{restart}.

exact line searches to a quadratic. In this case,

$$g_i^{\mathrm{T}} H_r y_i = g_i^{\mathrm{T}}\left(H - \frac{s_r y_r^{\mathrm{T}} H}{s_r^{\mathrm{T}} y_r}\right) y_i = g_i^{\mathrm{T}} H y_i - (g_i^{\mathrm{T}} s_r)\frac{y_r^{\mathrm{T}} H y_i}{s_r^{\mathrm{T}} y_r} = g_i^{\mathrm{T}} H y_i,$$

so the formulae in (7) and (8) are the same. Hence Beale's recurrence is another way of writing a preconditioned CG algorithm. Indeed we will henceforth restrict our attention primarily to preconditioned CG algorithms with the understanding that our discussion includes Beale's method.

Our discussion now returns to the matrix Q_i introduced by Perry and to Shanno's idea of what we call the 'padding' of Q_i. But it is best that we begin with some discussion. Consider the quasi-Newton update formula (1). In the standard quasi-Newton method, a search direction is computed as $d_{i+1} = -H^* g_i$. In this computation, there are a number of terms which disappear in the quadratic case, namely those with the term $s_i^{\mathrm{T}} g_i$, which is 0. Since a quasi-Newton algorithm effectively replaces f with a local quadratic approximation, one might therefore suggest that quasi-Newton updates should be modified by explicitly removing all terms in H^* ending in 's_i', but we believe that such a suggestion would not be well-received for it relies far too heavily on the behaviour of a quadratic model and the inherent assumption of exact line searches. It is our contention, in the context of conjugate gradient methods, that the same reasoning applies and that any such terms should not be eliminated. However, we suggest that precisely what has taken place, in an implicit way, ever since the introduction of conjugate gradient methods for minimization in 1963 [5], is that the 's_i' terms have been removed.

Consider the formulae (3) with $H = I$ so that Q_i in (4) is given by

$$Q_i = I - \frac{s_i y_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i}.$$

Perry [10] suggested modifying this matrix and wrote

$$\bar{Q}_i = I - \frac{s_i y_i^{\mathrm{T}} + s_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i}.$$

$\bar{Q}_i$ is not symmetric, but $y_i^{\mathrm{T}} \bar{Q}_i = s_i^{\mathrm{T}}$. Shanno [12] later modified and expanded this approach and, with further 'padding', wrote

$$H_i = I - \frac{s_i y_i^{\mathrm{T}} + y_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i} + \left(1 + \frac{y_i^{\mathrm{T}} y_i}{s_i^{\mathrm{T}} y_i}\right)\frac{s_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i}. \tag{9}$$

A major problem with $\bar{Q}_i$ as introduced by Perry is that it fails to satisfy the secant equation (i.e. $\bar{Q}_i y_i \neq s_i$). What Shanno saw is that further padding is essential, and from (9) it is clear that H_i is symmetric and $H_i y_i = s_i$. Furthermore, if an exact line search is used, the formulae $d_{i+1} = -Q_i g_i$ and $d_{i+1} = -H_i g_i$ generate identical directions. This supports our contention that some 's_i' terms have been implicitly and unknowingly dropped in CG methods. Also observe, as Shanno did, that H_i is

actually a quasi-Newton update matrix, namely

$$H_i = U(I, i),$$

of the fixed identity matrix.

For the preconditioned CG algorithm, the same steps can be followed. Pad, or resurrect if you like, the matrix Q_i in (4) to get

$$H_i = H - \frac{s_i y_i^{\mathrm{T}} H + H y_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i} + \left(1 + \frac{y_i^{\mathrm{T}} H y_i}{s_i^{\mathrm{T}} y_i}\right) \frac{s_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i}.$$

Clearly $H_i = U(H, i)$ and, replacing (3c) by $d_{i+1} = -H_i g_i$, we see that a preconditioned conjugate gradient algorithm should be interpreted as a quasi-Newton algorithm in which a fixed matrix H is updated at each step.

In the particular case of Beale's recurrence, we noted that it has the preconditioner H_r so that $H_i = U(H_r, i)$. Furthermore, the formula for H_r may be padded in an identical fashion, whence

$$H_r \equiv \dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H}}{s_r^{\mathrm{T}} y_r}$$

becomes

$$H_1 = \dot{H} - \frac{s_r y_r^{\mathrm{T}} \dot{H} + \dot{H} y_r s_r^{\mathrm{T}}}{s_r^{\mathrm{T}} y_r} + \left(1 + \frac{y_r^{\mathrm{T}} \dot{H} y_r}{s_r^{\mathrm{T}} y_r}\right) \frac{s_r s_r^{\mathrm{T}}}{s_r^{\mathrm{T}} y_r} = U(\dot{H}, r).$$

Thus we will write Beale's recurrence as

$$d_{r+i} = -H_i g_{r+i-1}, \quad i \geq 1, \tag{10a}$$

with

$$H_1 = U(\dot{H}, r), \tag{10b}$$

$$H_i = U(H_1, r+i-1), \quad i > 1. \tag{10c}$$

3. Multiple updates: The VSCG algorithm

The preceding discussion has left one point open: how should the preconditioner H be constructed? Here we will present a strategy based on successive updating with a quasi-Newton formula which will result in a straightforward manner in a CG algorithm which can use a variable amount of storage. Furthermore, the algorithm will reduce to Shanno's 'memoryless quasi-Newton algorithm' in one instance. Details of the algorithm will be left to the next section.

Suppose that we begin at x_r, which we will term a restart point, and suppose that x_r was reached along a descent direction d_r from x_{r-1} in such a way that $s_r^{\mathrm{T}} y_r > 0$. Algorithm VSCG then consists of two parts, as follows.

QN-part: Choose a positive definite matrix H_0. Iterate for $i = 1, 2, \ldots, m$:

$$H_i = U(H_{i-1}, r+i-1), \tag{11a}$$

$$d_{r+i} = -H_i g_{r+i-1}, \tag{11b}$$

$$x_{r+i} = x_{r+i-1} + \alpha_{r+i} d_{r+i}.$$

CG-part: From the point x_{r+m} reached by the QN-part, and using the fixed matrix H_m as preconditioner, iterate for $i = m+1, m+2, \ldots$:

$$H_i = U(H_m, r+i-1), \tag{12a}$$

$$d_{r+i} = -H_i g_{r+i-1}, \tag{12b}$$

$$x_{r+i} = x_{r+i-1} + \alpha_{r+i} d_{r+i}.$$

As can be seen, the algorithm VSCG is essentially very simple. The two parts differ only in the definition of H_i: for the first m steps, H_i is an update of the previous matrix H_{i-1}; subsequently, it is an update of a fixed matrix H_m. Thus the QN-part can be viewed as constructing an appropriate preconditioner H_m, and the CG-part is the implementation of a preconditioned conjugate gradient algorithm in the form described in Section 2. The choice of H_0 is left to the next section. The significance of m and storage of $H_1, \ldots, H_m$ will be given soon.

For simplicity in the presentation, we will in certain instances assume $r = 1$; indeed in the implementation of the algorithm, each restart point is defined to be x_1. The eqs. (11) and (12) are then a bit simpler, and we will sometimes use them in this form:

$$H_i = U(H_{i-1}, i) \quad \text{or} \quad H_i = U(H_m, i)$$

and

$$d_{i+1} = -H_i g_i.$$

Algorithm VSCG has some important properties. First, one may ensure that each H_i is positive definite by ensuring that $s_i^T y_i > 0$; this is a well known property of BFGS updates. It is then clear from (11b) and (12b) that all directions are descent directions. Therefore, in contrast to other CG algorithms, it is easy to obtain a downhill direction at each step, for the condition $s_i^T y_i > 0$ is easy to ensure. When f is quadratic, finite termination is obtained, provided n steps are taken from x_r without restarting. This follows immediately from a theorem given in Buckley [2]. Of course, this statement is only true if each line search is exact, whereas, in the algorithm as implemented, not all line searches will be exact, so finite termination is actually lost. This is not a weakness of the method, for, as in quasi-Newton methods, insistence on finite termination for quadratics will degrade performance on general f. Our numerical results in Section 6 concerning quadratic interpolation will support this claim.

Now consider how $H_1, \ldots, H_m$ can be stored. One way has been discussed in [1], and we will repeat it here for completeness since the details are a bit different. Notice that the matrices H_i are not themselves needed; only products of the form $H_i v$ for $v \in \mathbb{R}^n$ are required. Consider therefore the equation derived from (11a) and (12a):

$$\begin{aligned} H_i v &= H_q v - \left[\frac{u_i^{\mathrm{T}} v}{\eta_i} - \left(1 + \frac{\nu_i}{\eta_i}\right)\frac{s_i^{\mathrm{T}} v}{\eta_i}\right] s_i - \frac{s_i^{\mathrm{T}} v}{\eta_i} u_i \\ &= H_q v - \sigma_i s_i - \mu_i u_i, \end{aligned} \tag{13}$$

where $\nu_i = y_i^{\mathrm{T}} H_q y_i$, $\eta_i = s_i^{\mathrm{T}} y_i$ and $u_i = H_q y_i$. Here H_q is either H_{i-1} or H_m according to whether H_i is defined by (11a) or (12a); i.e. $q = i-1$ or $q = m$. The iteration (11) or (12) requires computation of $H_i v$ with $v = g_i$. This is done using (13) by computing primarily

(i) $H_q g_i$;
(ii) $u_i = H_q y_i$, $\nu_i = y_i^{\mathrm{T}} u_i$ and $\eta_i = s_i^{\mathrm{T}} y_i$; and
(iii) $u_i^{\mathrm{T}} g_i$, $s_i^{\mathrm{T}} g_i$, $\sigma_i s_i$ and $\mu_i u_i$.

In (i) and (ii), computation of $H_q g_i$ or $H_q y_i$ is straightforward by applying (13) recursively. For example,

$$\begin{aligned} H_q v &= H_0 v - \sum_{j=1}^{q} \left\{\left[\frac{u_j^{\mathrm{T}} v}{\eta_j} - \left(1 + \frac{\nu_j}{\eta_j}\right)\frac{s_j^{\mathrm{T}} v}{\eta_j}\right] s_j - \frac{s_j^{\mathrm{T}} v}{\eta_j} u_j\right\} \\ &= H_0 v - \sum_{j=1}^{q} \{\sigma_j s_j + \mu_j u_j\}. \end{aligned}$$

Provided we have stored ν_j, η_j, u_j and s_j for $j = 1, \ldots, q$, this can easily be computed, with the main work done in forming $u_j^{\mathrm{T}} v$, $s_j^{\mathrm{T}} v$, $\sigma_j s_j$ and $\mu_j u_j$; the operation count for calculating $H_q v$ is $q(4n)$, ignoring $H_0 v$.

Thus, for one iteration, the operation counts are:

(i) $u_i = H_q y_i$: $4qn$;
(ii) $\nu_i = u_i^{\mathrm{T}} y_i$ and $\eta_i = s_i^{\mathrm{T}} y_i$: $2n$;
(iii) $H_q g_i$: $4qn$;
(iv) $u_i^{\mathrm{T}} g_i$, $s_i^{\mathrm{T}} g_i$, $\sigma_i s_i$ and $\mu_i u_i$: $4n$;

hence the total is either $(8i-2)n$ or $(8m+6)n$. Provided $m \ll n$, this is in either case only a moderate multiple of n. Remember that, when $q = i-1$, we must also save ν_i, η_i, s_i and u_i for the next step; when $q = m$ they are discarded. The total storage needed for $H_1, \ldots, H_m$ is then $m(2n+2)$ locations. Note that an alternative to this method of handling the matrices is the product form of H_i described by Nocedal [8].

To conclude this section, consider the special cases $m = 1$ and $m = 2$. If we choose $m = 1$, VSCG reduces to Shanno's 'memoryless quasi-Newton algorithm' MQN which is implemented in the program CONMIN [14]. When $m = 1$, we have $d_{r+1} = -H_1 g_r$ where $H_1 = U(I, r)$; subsequently for $i > 1$, $d_{r+i} = -H_i g_{r+i-1}$ where $H_i = U(H_1, r+i-1)$. This is precisely Beale's algorithm with $\dot{H} = I$ as direct comparison with eqs. (10) shows. (At least, as before, this is precisely Beale's algorithm

on quadratics with exact searches; alternately this can be viewed as a modified implementation of Beale's algorithm.) In contrast, Shanno's method of deriving MQN was to take the Beale recurrence and directly add the necessary padding terms. We believe that our approach has the advantage of a certain clarity and simplicity, and that it emphasizes the relationship of MQN to preconditioning. It also makes the extension to a variable storage algorithm easy.

For $m = 2$, let us write $\dot{H}$ in place of H_1, and renumber $H_2, H_3, \ldots$ as $H_1, H_2, \ldots$. Then the updates of VSCG can be written as

$$\dot{H} = U(I, r),$$

$$H_1 = U(\dot{H}, r+1),$$

$$H_i = U(H_1, r+i) \quad \text{for } i > 1.$$

Comparison with (10) shows that this is in fact a preconditioned Beale restart algorithm, where technically the restart is at $r+1$ rather than r. The step from x_r to x_{r+1} builds the preconditioner $\dot{H}$ for Beale's algorithm.

4. Algorithmic details

There is, of course, a 'giant step' between the brief description of an algorithm, as in (11) and (12), and its implementation. Here we will discuss some of the points which we found essential to having a successful implementation. Briefly, we will consider (a) when to restart, i.e. when to declare that the current point is a restart point and to redefine r to have the current value of i; (b) how to define H_0 at the restart point and how to scale the search directions; and (c) how to select a line search subalgorithm. We will not be giving a detailed description of the code implementing VSCG. For readers so interested, the authors plan to publish the algorithm separately. However, in Section 6 we will give some numerical results concerned with the choice of strategy.

The question of finding a dynamic criterion for restarting was first considered by Powell [11], where he was dealing with the Beale recurrence in the form (5) with $H = I$. He suggested that r be reset to i when, at x_i,

$$\tau_i \equiv \frac{g_i^{\mathrm{T}} g_{i-1}}{g_{i-1}^{\mathrm{T}} g_{i-1}} \tag{14a}$$

was significantly different than 0, say when $|\tau_i| > \rho$, where typically ρ would be 0.2. This test accomplished two things. First, in the quadratic case $g_i^{\mathrm{T}} g_{i-1}$ would always be zero, no restarts would take place, and finite termination could occur. Thus, a measurable departure from 0 for τ_i would indicate strong local nonquadratic behaviour and hence would be indicative of a need for restarting. Second, as Powell observed, when using Beale's recurrence in the nonquadratic case, convergence can occur to a point at which the gradient is nonzero. This would cause τ_i to be near 1, which would force a restart and presumably avoid the false convergence.

For the algorithm VSCG, a similar criterion is suggested. For the preconditioned CG algorithm (3) with preconditioner H_m as in (12), the ratio (14a) becomes

$$\bar{\tau}_i = \frac{g_i^{\mathrm{T}} H_m g_{i-1}}{g_{i-1}^{\mathrm{T}} H_m g_{i-1}}. \tag{14b}$$

Therefore an obvious suggestion is to restart when $|\bar{\tau}_i| > \rho$. On the other hand, for quadratic f and with $H_0 = I$, the results in Buckley [2] show that the algorithm (11), (12) will generate the same set of points as would be generated by the simple CG algorithm (3) with $H = I$. In particular, using Theorem 1 of [2], we see that $\bar{\tau}_i = \tau_i$. This suggests that there is no need to implement the computationally more expensive test with $\bar{\tau}_i$ even in the nonquadratic case, in which τ_i and $\bar{\tau}_i$ will be expected to be different. In Section 6 we will compare the effect of using restart tests based on the two quantities τ_i and $\bar{\tau}_i$, and we will see that it is indeed sufficient to use τ_i.

The scaling problem, i.e. in part the choice of H_0, was discussed by Shanno in conjunction with the MQN algorithm. His conclusions, appropriately interpreted, apply here. First, for quasi-Newton algorithms, the experience of Shanno and Phua [13] strongly favored the use of the Oren–Spedicato scaling [9] of the BFGS update formula, with the proviso that the scaling is most successful if it is only applied to the initial update. For the algorithm MQN, Shanno treated it as a sequence of quasi-Newton steps (in which a fixed matrix is being updated at most steps). Applying their earlier conclusions, Shanno decided that the correct way to implement MQN would be to use the Oren–Spedicato scaled version of the BFGS update for the first step, and then to continue with the normal BFGS update; numerical evidence was presented to support the decision. Suppose we write the Oren–Spedicato scaled BFGS update as

$$\begin{aligned} H^* &= \left(H - \frac{s_i y_i^{\mathrm{T}} H + H y_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i} + \frac{y_i^{\mathrm{T}} H y_i}{s_i^{\mathrm{T}} y_i} \frac{s_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i} \right) \gamma_i + \frac{s_i s_i^{\mathrm{T}}}{s_i^{\mathrm{T}} y_i} \\ &= U(\gamma_i H, i) \equiv \mathrm{OS}(H, i), \end{aligned}$$

where $\gamma_i = \eta_i / \nu_i$ and η_i and ν_i are as in (13). Then, using our description of MQN, Shanno's strategy is simply stated by requiring that

$$H_1 = \mathrm{OS}(I, r),$$

$$H_i = U(H_1, r + i - 1) \quad \text{for } i > 1.$$

Note that we have in effect replaced the choice $H_0 = I$ with $H_0 = \gamma_r I$.

It is clear how to adapt this strategy to VSCG. The QN-part (11) consists of a sequence of quasi-Newton steps; the Shanno–Phua results suggest that the first of these in (11a) be altered to $H_1 = \mathrm{OS}(I, r)$; the remainder are left unchanged. The CG-part (12) can again be interpreted as a continuing sequence of quasi-Newton steps (based on a fixed update) so that one should *not* replace $H_i = U(H_m, i)$ with $H_i = \mathrm{OS}(H_m, i)$. Alternatively, it could be argued that the Oren–Spedicato update only applies to quasi-Newton algorithms and is therefore not appropriate for the

CG-part of the algorithm. In any event, at each restart the initial direction d_{r+1} (and it alone) is rescaled with the Oren–Spedicato formula.

The question of scaling the search directions other than d_{r+1} is still open. It is known that one of the failings of CG algorithms is that they have difficulty determining a suitable length for each direction d_{i+1}. This occurs because they have no 'memory' of previous steps. For Shanno's MQN, all steps other than a restart step are CG steps. Therefore, according to [12], Shanno scaled each subsequent direction using the formula Fletcher [6] introduced with his CG algorithm:

$$\alpha_{i+1} = 2\frac{f_{i-1} - f_i}{g_i^{\mathrm{T}} s_i}\alpha_i. \tag{15a}$$

Because the updates (12) involve only the total step s_i and not the actual direction d_i, the scaling of the direction d_{i+1} amounts simply to the initial choice for α_{i+1}, as given. Alternately, one may choose α_{i+1} as Shanno actually did in the published code CONMIN [14]:

$$\alpha_{i+1} = \alpha_i \times \frac{d_{i+1}^{\mathrm{T}} g_i}{d_i^{\mathrm{T}} g_{i-1}}. \tag{15b}$$

Experience showed that these worked well, both for Fletcher and for Shanno. For VSCG, there are both quasi-Newton and CG nonrestart steps (except when $m = 1$). For the QN-part, we proceed exactly as we would for a quasi-Newton algorithm, for after all, that is exactly what it is until the storage limit is reached. Therefore each direction d_{i+1} is accepted as computed by (11b). It is not rescaled in length, which is reasonable since these are quasi-Newton steps, which do have 'memory' and hence determine a reasonable length for d_{i+1}. For the CG-part, we proceed as in a CG algorithm; each search direction is computed according to (12b) and then an initial choice for α_{i+1} is made according to one of the formulae of (15). Results in the next section indicate that the strategies which we have chosen work acceptably in practice.

Finally, we consider the line search. Essentially we have the question: since the algorithm is in many ways intermediate between a quasi-Newton method and a conjugate gradient method, which kind of search should we use? The answer is, not surprisingly, intermediate, at least based on computational experience. As in the discussion of scaling, our answer is to some extent, but not entirely, to use a quasi-Newton strategy for the QN-part and a conjugate gradient strategy for the CG-part.

The basic point to a quasi-Newton search is that no true 'search' is required. In order to keep the updates positive definite, it is necessary to ensure that $s_i^{\mathrm{T}} y_i > 0$ at each step, but that is a mild restriction. An important property of quasi-Newton algorithms is that, near the solution, the choice $\alpha_i = 1$ may be taken for each step (with only the proviso $s_i^{\mathrm{T}} y_i > 0$) and that superlinear convergence results. Therefore the quasi-Newton strategy is generally to try $\alpha_i = 1$, test the condition $s_i^{\mathrm{T}} y_i > 0$ at

the new point, accept the new point as x_i if the test is satisfied, and only apply a search technique when the test fails. As already stated, the QN-part of VSCG is indeed a quasi-Newton algorithm, so the value of α_i is chosen essentially as described. (Here, as elsewhere, the actual details are a bit more involved, but those can be seen in the published algorithm.)

For the CG-part of VSCG, more is required. It is generally acknowledged that a true line search is required for conjugate gradient algorithms such as (3) or (5). A procedure must be given which produces a point x_i such that the exact line search criterion $d_i^T g_i = 0$ is 'nearly' satisfied. In particular, to ensure that the next direction d_{i+1} is downhill, the equation (3c) implies that x_i must be chosen so that

$$\frac{d_{i-1}^T g_i}{d_{i-1}^T y_i} < \frac{g_i^T H g_{i-1}}{g_{i-1}^T H g_{i-1}}. \tag{16}$$

Experience has shown that in fact condition (16) is not strong enough; a strengthened form of (16) which forces the point x_i to be even nearer to a local minimum along d_i is usually imposed. The result is that, for algorithms such as (3) or (5), the ratio of the number of function evaluations to the number of iterations is fairly large, say at least 2 and more typically 2.5 to 3. One point in favor of these line searches, though, is that the search procedure always uses some form of quadratic or cubic interpolation, with the result that, in the quadratic case, line searches are exact and finite termination is obtained. That is not true for quasi-Newton algorithms which often accept $\alpha_i = 1$.

But now consider specifically the CG-part of VSCG. The most important point is to observe that the search direction d_{i+1} in (12b) is in the form $d_{i+1} = -H_i g_i$. Thus all that is needed to ensure that d_{i+1} is downhill is that $s_i^T y_i > 0$, for that guarantees that H_i is positive definite. As observed, that is in definite contrast to algorithm (3). (In fact the contrast is even greater for (5), for it may not even be possible to choose α_i to make d_{i+1} in (5d) downhill.) One might therefore be tempted to conclude that we may now proceed in the same way as in the QN-part and accept $\alpha_i = 1$ for the most part. Computations have shown that this does not work well. The reason is that the CG-part is not a true quasi-Newton algorithm (since a fixed matrix is being updated), and there are therefore no results which suggest that $\alpha_i = 1$ is an acceptable choice, even near the solution. Our compromise is to follow the same strategy as in the QN-part, except that α_i will not be accepted unless at least one quadratic interpolation has taken place along that line. We have found this to work quite successfully, and indeed in one case, VSCG is much better than a quasi-Newton algorithm, a fact which we attribute to the presence of the forced quadratic interpolation in the CG-part.

We would also like to make an observation on a minor point which will be investigated in Section 6. If one examines formulae (12), it is easily seen that the step with $i = m + 1$, i.e. the first step of the CG-part, is in fact a quasi-Newton step. Indeed, it is not until we begin the computation of H_{m+2} leading to d_{r+m+2} that we can detect that the CG-part is different from the QN-part, at least as far as the

description (11), (12) is concerned. What this does indicate though, is that one must be careful about the step along d_{r+m+1} from x_{r+m} to x_{r+m+1}. Should it be considered as a quasi-Newton step or as a conjugate gradient step? As we have seen, that definitely affects the line search strategy. We will discuss this point in Section 6, but let us say here that it seems quite important to view the step along d_{r+m+1} as a conjugate gradient step, especially when m is small.

We conclude this section with a summary of the algorithm as we feel it should be implemented. Many of the actual implementation details will be glossed over. Assume m is given. As the previous paragraph noted, the algorithm sometimes distinguishes between being in the CG- or QN-part, and whether a line search implements a CG or QN strategy. One important point that this description will not indicate explicitly is that we do require that each iteration will reduce the function value.

1. Cold-start. Given x_0, set $H_0 = I$ and compute $f_0 = f(x_0)$ and $g_0 = g(x_0)$. Terminate if x_0 is close enough to a local minimum. Set $i = 1$ and compute $d_1 = -H_0 g_0$. Note that, since x_1 is always treated as a restart point, the initial step is considered to be the final step in the CG-part. Some minor adjustments are needed in what follows for the initial step.

2. Step from x_{i-1} along d_i.

(a) Choose an initial value for α_i: if this is a QN-step, set $\alpha_i = 1$; otherwise scale the previous value α_{i-1} as in Shanno and Phua [12, 13].

(b) Do a line search along d_i. The important points are:

(i) If this is a QN-step: Check if $s_i^T y_i > 0$ at $x_{i-1} + \alpha_i d_i$. If so, the line search is complete. If not, determine a new value for α_i (usually by interpolation) and repeat.

(ii) If this is not a QN-step: Check if $s_i^T y_i > 0$ at $x_{i-1} + \alpha_i d_i$. If so, and if an interpolation has been done at least once during this line search, then the line search is complete. If so, but no interpolation has yet been done, then force an interpolation and repeat. If $s_i^T y_i \leq 0$, then determine a new value for α_i (usually by interpolation) and repeat.

(c) Set $x_i = x_{i-1} + \alpha_i d_i$ and check for termination.

3. Declare x_i to be a restart point if this is in the CG-part of the algorithm and if either $|g_i^T g_{i-1}| > 0.2\|g_{i-1}\|$ or n steps have been taken from the last restart point.

4. Compute H_i and d_{i+1}:

(a) If this is a restart step, then compute y_i, $\nu_i = y_i^T y_i$ and $\eta_i = s_i^T y_i$, set $u_i = y_i$ and store s_i, u_i, ν_i and η_i to define H_1. Reset i to 1, and compute $d_2 = -H_1 g_1$.

(b) If not, then set $q = i - 1$ if this is in the QN-part, or set $q = m$ if this is in the CG-part. Then compute y_i, $u_i = H_q y_i$, $\nu_i = u_i^T y_i$ and $\eta_i = s_i^T y_i$ to define H_i. If this is in the QN-part, store s_i, u_i, ν_i and η_i so as to keep H_i. Then calculate $s_i^T g_i$, $u_i^T g_i$ and $H_q g_i$ in order to determine $d_{i+1} = -H_i g_i$ according to (13). Increment i by 1.

5. Repeat from 2.

5. Numerical results

In this section and the next we wish to consider two points. First, we will give numerical results that show that we have met our main objective, namely, that we can minimize more efficiently by increasing the amount of storage available to this conjugate gradient routine. Then, in Section 6, we will investigate some of the choices one has when actually implementing the algorithm.

We begin with some words about our testing method. All test results quoted here were obtained using 64-bit double precision on a VAX 11/780. Termination occurred with $\|g\| < 10^{-3} \times \max(1, \|x\|)$, as in [14]. We have used the general purpose testing package described in [4] to produce our basic results; the tables appearing here are summaries of those produced by that package. For purposes of comparison, we have used an equivalent to Shanno's CONMIN code [14]. This is partly because CONMIN can operate either as a conjugate gradient algorithm or as a quasi-Newton algorithm. The quasi-Newton part is robust and probably as effective as any unconstrained quasi-Newton code generally available; the conjugate gradient part is the one referred to in this paper as MQN and it is also highly regarded. Therefore we feel that the CONMIN code provides a good up-to-date basis for comparison. The results were actually produced by the code called VSCG. However, with one update, VSCG produces results which are identical to those obtained from CONMIN, which is what one would hope since Shanno's algorithm is a special case of our algorithm, as explained. In addition, VSCG has imbedded in it the same quasi-Newton code as is found in CONMIN. Therefore in what follows a reference to CONMIN means the equivalent code in VSCG.

The test functions we have chosen are fully documented in [4] and are in two groups. We summarize the second group of them here. These are chosen because they are of moderate dimension and readily available, but still not unreasonably expensive to use for comparative testing. We believe that this second test set is sufficient to be able to support our basic claim. These test functions are

PWSING, variable n:

$$f(x) = \sum_{i=1}^{n/4} [(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4].$$

NONDIA, variable n:

$$f(x) = \sum_{i=2}^{n} [100(x_1 - x_i^2)^2 + (1 - x_i)^2].$$

TRIDIA, variable n:

$$f(x) = \sum_{i=2}^{n} [i(2x_i - x_{i-1})^2].$$

EXTROS, variable n:

$$f(x) = \sum_{i=1}^{n/2} [100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2].$$

CHAROS, variable n:

$$f(x) = \sum_{i=2}^{n} [4\delta_i(x_{i-1} - x_i^2)^2 + (1 - x_i)^2].$$

The constants δ_i are available from [4] or from Toint [15].

To begin our evaluation of the new code, consider the results in Table 1. Here the test set consists of a number of standard test problems, all of rather small dimension. Of course, one can point out that the purpose of the algorithm which we are describing is to handle problems of moderate or even large size, so it does not seem reasonable to exhibit test results for small problems. That is true. However, we feel that, at the very least, the new algorithm should not be any worse than standard algorithms on standard problems. The results in Table 1 demonstrate that this is indeed the case. This table shows the results of minimizing a collection of problems, first with the one-update conjugate gradient code as implemented in VSCG, then with the m-update algorithm of this paper, where m is successively 2, 4, 8 and 10, and finally with the quasi-Newton part of CONMIN – VSCG. Because the problems are small, there are some fluctuations in the number of function evaluations (see FUNCS) needed to minimize a particular function in the set. There is however a reasonably consistent trend for that number to decrease as m increases. Observe that for $m = 8$, VSCG is nearly as good as the QN-code, although for $m = 10$, the results become a bit poorer; however one should be cautious about attempting to read too much into such figures, for they really represent partly random variations.

Now consider the results in Table 2. These are analogous to Table 1, except that the set of test functions is different. Here all of them are of moderate dimension. Again we observe the rather consistent improvement in performance with increasing m.

Of course, we should state just what we mean when we say that the performance has improved. Primarily, we are looking at function evaluations, and it is the total of this figure for each value of m which shows most clearly the reduction in work. However, we should also point out that the total number of iterations (ITERS), the term RATIO (which is FUNCS/ITERS), and the figures MSECS and FSECS in Tables 3 and 4 are also interesting. The time spent in evaluation of the functions is in FSECS and the total time taken for each minimization is in MSECS; times are in milliseconds.

First consider Table 3 where the problems are the same small ones as in Table 1. As m changes, there is not a lot of change in the figure MSECS, for as m increases, there is a slight reduction in the number of function evaluations, but that

is balanced by some increase in the number of iterations and the overhead per iteration, which is $O(mn)$ for VSCG. Also, since n is small, the expressions $O(n)$ and $O(n^2)$ have little meaning, so the time for the QN-case is also not much different. But in Table 4, where n and n^2 are noticeably different, the situation changes. Now the overhead often dominates the time taken in evaluating the function. As m increases, the function count decreases. This reduces FSECS but has little effect on MSECS. Since the ratio FUNCS/ITERS is reduced steadily as m grows (because of how we implement the line search), the number of iterations does not decrease, but is steady, or may increase slightly. Also the overhead is $O(mn)$ per iteration. Thus, MSECS is also fairly steady or grows slightly. But the quasi-Newton results are quite different. The function count is indeed the least, but RATIO is near to 1, so ITERS is comparatively large. Therefore there are many iterations, and the overhead for each is $O(n^2)$. The result is that MSECS is far larger than with any of the conjugate gradient runs. The figure TIME – RATIO, which is MSECS/FSECS, also emphasizes the difference.

Thus, with VSCG we have in some sense the best of both worlds: if function evaluations are cheap, VSCG will do well because it has relatively small overhead; if function evaluations are expensive, VSCG will still do reasonably well because its function count is close to that of a quasi-Newton algorithm. Therefore we feel that we may conclude that VSCG successfully meets the primary objective which we stated at the beginning of this paper.

6. Tests on implementation strategies

In Section 4, we discussed certain points relating to the actual implementation of the algorithm. Here we will present numerical results which demonstrate how different strategies affect performance, and we will explain how we have chosen the strategy which was summarized at the end of Section 4 and used to obtain the results of Section 5 and why we feel that our choice is justified.

Table 5 contains a summary of a substantial number of tests. Each line of the table was obtained from tables similar to Tables 1 to 4; for obvious reasons of space the 15 tables are not reproduced here. The entries in the tables have the following meanings; for ease of reference for those who might be interested, we use the same names here as are used for the control variables in the code.

FIRST 1. When this is true, the first step in the CG-part of the algorithm (along d_{r+m+1}) is treated as a QN-step. This affects the initial choice of α_i, which is either 1 for a QN-step, or scaled according to (15) for a CG-step. In the ensuing discussion we will carefully distinguish between CG-steps and the CG-part. Steps after leaving x_{r+m} are always in the CG-part, but the first of these may or may not be treated as a CG-step.

ALPIS 1. When this is true, all steps begin with an initial choice of 1 for α_i, both in the QN-part and in the CG-part.

Table 1
FUNCS/ITERS for small problems

function	dimension	$m = 1$ CONMIN	$m = 2$	$m = 4$	$m = 8$	$m = 10$	$m = \infty$ QN
ROSENBRK	2	64/28	63/25	46/28	44/33	47/36	44/36
WHLST 3	2	49/18	37/20	31/17	29/24	32/26	40/28
HIMM 28	2	15/6	11/6	11/7	10/7	10/7	10/7
HIMLN 3	2	11/5	9/5	8/5	8/6	8/6	7/6
BEAL 58	2	26/10	26/13	21/15	16/12	17/13	15/13
ENGVL 1	2	20/8	20/11	14/9	15/11	15/11	14/13
SCHMVT	3	22/10	23/11	22/14	18/13	23/19	18/15
ENGVL 2	3	70/33	63/32	65/32	52/36	49/36	31/27
BARD 70	3	29/14	26/13	24/15	22/16	22/16	26/25
CRGLVY	4	54/24	59/30	43/28	35/28	39/27	56/54
BIGGS 6	6	111/53	77/41	95/59	61/41	49/39	45/41
BOX 663	3	25/10	33/16	35/19	36/21	37/26	40/32
PWSING	4	97/41	68/36	55/32	46/31	58/45	41/40
TOTAL		593/260	515/259	470/280	392/279	406/307	387/337
RATIO		2.28	1.99	1.68	1.41	1.32	1.15

Table 2
FUNCS/ITERS for larger problems

function	dimension	$m = 1$ CONMIN	$m = 2$	$m = 4$	$m = 8$	$m = 10$	$m = \infty$ QN
PWSING 60	60	131/62	76/41	64/36	71/45	44/33	54/52
PWSING 80	80	73/36	87/47	57/33	40/31	70/54	57/56
NONDIA 10	10	78/24	54/26	44/26	43/31	45/32	46/33
NONDIA 20	20	72/24	46/21	48/28	43/29	41/28	44/32
NONDIA 30	30	64/26	45/26	43/27	42/32	41/29	44/32
TRIDIA 10	10	19/9	35/18	33/20	33/23	30/25	17/16
TRIDIA 20	20	39/19	40/20	60/34	59/44	53/38	29/28
TRIDIA 30	30	61/30	63/31	75/43	63/38	67/42	37/36
EXTROS 10	10	61/27	62/28	46/28	44/33	47/36	44/36
EXTROS 20	20	61/27	62/28	46/28	44/33	47/36	44/36
EXROS 10A	10	47/17	37/20	33/19	34/23	34/21	43/30
CHAROS 10	10	73/34	61/34	51/34	50/37	52/42	39/37
CHAROS 25	25	103/49	106/56	70/42	63/41	61/42	55/51
TOTAL		882/384	774/396	670/398	629/440	632/458	553/475
RATIO		2.30	1.95	1.68	1.43	1.38	1.16

QUADIN. In certain instances, a line search is not deemed complete unless at least one interpolation has taken place. As indicated in Section 4, that is a strategy generally thought appropriate for a conjugate gradient algorithm. Here we can control the level at which interpolations are forced. If

QUADIN = 0: an interpolation is forced for any CG-step, where the identification of a CG-step depends on FIRST 1:

Table 3
MSECS/FSECS for small problems

function	dimension	$m = 1$ CONMIN	$m = 2$	$m = 4$	$m = 8$	$m = 10$	$m = \infty$ QN
ROSENBRK	2	152/23	137/23	117/31	168/55	180/23	113/31
WHLST 3	2	90/12	90/43	74/20	109/20	121/20	74/16
HIMM 28	2	20/1	8/0	39/8	20/1	20/1	31/8
HIMLN 3	2	20/1	20/8	31/20	27/1	31/1	20/12
BEAL 58	2	51/31	70/12	63/8	63/12	59/1	47/16
ENGVL 1	2	39/1	51/16	51/8	43/23	63/12	20/1
SCHMVT	3	59/20	59/31	78/31	82/23	121/12	63/8
ENGVL 2	3	180/43	160/63	180/51	211/63	211/39	102/27
BARD 70	3	117/51	78/59	109/43	121/43	121/35	109/39
CRGLVY	4	137/66	184/43	160/43	180/27	199/47	195/51
BIGGS 6	6	1164/945	852/625	1109/801	762/500	656/367	523/375
BOX 663	3	141/133	203/152	238/168	250/141	285/148	254/168
PWSING	4	227/86	156/78	188/63	199/59	301/20	141/20
TOTAL		2395/1410	2066/1152	2438/1293	2234/965	2367/723	1691/770
TIME-RATIO		1.70	1.79	1.89	2.32	3.27	2.20

Table 4
MSECS/FSECS for larger problems

function	dimension	$m = 1$ CONMIN	$m = 2$	$m = 4$	$m = 8$	$m = 10$	$m = \infty$ QN
PWSING 60	60	1762/441	1211/262	1309/164	1965/254	1609/148	8184/289
PWSING 80	80	1332/301	1809/320	1500/191	1699/207	3504/289	15973/207
NONDIA 10	10	301/94	219/94	250/47	305/66	383/74	379/66
NONDIA 20	20	383/105	258/98	402/63	488/74	531/117	719/86
NONDIA 30	30	465/188	430/145	578/125	727/55	781/105	1473/148
TRIDIA 10	10	82/20	164/63	277/27	230/20	270/59	129/12
TRIDIA 20	20	270/74	426/121	570/113	797/156	793/125	609/47
TRIDIA 30	30	590/121	691/207	1063/195	1602/164	1559/223	1531/105
EXTROS 10	10	223/78	250/39	219/43	316/74	383/31	313/35
EXTROS 20	20	316/102	359/94	504/145	523/59	594/74	723/74
EXROS 10A	10	148/39	129/35	160/55	250/23	223/63	270/59
CHAROS 10	10	328/145	352/137	340/102	430/86	504/78	297/63
CHAROS 25	25	926/355	1102/313	879/211	1141/207	1211/188	1680/180
TOTAL		7125/2063	7398/1926	8051/1480	10473/1445	12344/1574	32277/1371
TIME-RATIO		3.45	3.84	5.44	7.25	7.84	23.54

QUADIN = 1: an interpolation is forced for any step in the CG-part, so that FIRST 1 has no effect;

QUADIN = 2: no quadratic interpolations are forced at all; and if

QUADIN = 3: an interpolation is forced on every line search, including those in the QN-part.

Table 5
Function evaluation totals

FIRST 1	ALPIS 1	QUADIN	HTEST	function counts for m = 1	2	4	8	10	TOTAL	
FALSE	FALSE	1	FALSE	882	774	670	629	632	4998	row 1
TRUE	FALSE	1	FALSE	870	739	709	619	640	4983	row 2
FALSE	FALSE	1	TRUE	887	778	706	658	604	5042	row 3
FALSE	TRUE	1	FALSE	826	815	699	624	640	5025	row 4
TRUE	TRUE	1	FALSE	same as row 4						
TRUE	FALSE	0	FALSE	984	865	725	649	607	5467	row 6
FALSE	FALSE	0	FALSE	same as row 1						
FALSE	FALSE	2	FALSE	1038	948	699	636	593	5494	row 8
TRUE	FALSE	2	FALSE	966	841	763	633	625	5179	row 9
TRUE	TRUE	2	FALSE	926	896	804	632	638	5270	row 10
FALSE	TRUE	2	FALSE	same as row 10						
FALSE	FALSE	3	FALSE	882	828	739	715	734	5458	row 12
TRUE	FALSE	3	FALSE	870	887	770	733	739	5478	row 13
TRUE	TRUE	3	FALSE	826	830	756	721	741	5342	row 14
FALSE	TRUE	3	FALSE	same as row 14						

HTEST. When this is true, the restart criterion is based on the test (14b) using the H-metric; otherwise it is based on the original Powell test (14a).

TOTAL. The columns headed 1, 2, 4, 8 and 10 contain the function evaluation totals obtained from tables similar to Tables 1 to 4. These entries are summed by row and the sum put in the column named TOTAL. Thus, for each setting of the parameters, we have a measure of the total work needed to solve a reasonable block of problems.

Now we consider the results in Table 5. We have included some rows in the table which are redundant. They are there to emphasize that with certain settings of QUADIN and ALPIS 1, the parameter FIRST 1 has no effect. The descriptions of these variables given earlier should make it clear why this is so, but the extra lines are still included to avoid any possible confusion.

Initially, note that the results in the first five rows are consistently better than those in the remainder. It is no coincidence that QUADIN = 1 for the better results. Instead, it suggests that the decision regarding interpolation in the line searches is important.

Now, to be specific, let us compare rows 1 and 2 of Table 5. Both force a quadratic interpolation throughout the CG-part; the difference is only whether or not the step along d_{r+m+1} begins with $\alpha_i = 1$ or with a scaled value according to (15). It appears that this is not a sensitive issue, since the results are similar.

Next, compare rows 1 and 3. The question here is whether we should implement restarts according to (14a) or (14b). Again, the difference is slight. To implement the test (14b) is, however, a bit more expensive. It requires either the storage of

an additional vector or the computation of an additional matrix-vector product (a point which is best understood by examining the code for VSCG). We chose the additional computation because it is most easily made optional in the code; however the results indicate that the extra computation gains little, so all further tests are done with the simpler restarting criterion (14a).

In row 4, we consider the possibility of allowing α_i to be taken initially as 1 in all steps. This seems to have little detrimental effect; that is perhaps because most of the steps taken by the algorithm belong in the QN-part anyway, so not many steps are affected. It also suggests that the mixing of the QN and CG algorithms in this manner results in conjugate gradient steps which are better scaled in length than is normally the case. However, it is perhaps equally true that this is simply a much less sensitive question than that of the interpolation.

Let us now consider different levels of forcing the quadratic interpolation. Take rows 2 and 6. The only difference is in the action taken on the first step in the CG-part, i.e. along d_{r+m+1}. For row 2, quadratic interpolation is forced along this step, while for row 6 it is not forced. This seems to have a significant effect. Actually, for the larger values of m, the effect is less, which is perhaps what one might expect since more of the steps are in the QN-part. But performance seems to be degraded when m is small; in other words, for Shanno's CONMIN or for VSCG with not many more updates, it does seem important that a quadratic interpolation be done on any step in the CG-part.

Let us now look at rows 8 to 10, where no quadratic interpolation is forced anywhere in the algorithm. For a quasi-Newton code, one would expect this to have no effect, and such is indeed the case for the larger values of m. But for small m, where the CG-steps form a more important part of the code, as already discussed in the previous paragraph, one would expect the interpolation to matter. Indeed it does. Clearly the degraded performance for QUADIN = 2 is primarily with $m = 1, 2$ or sometimes 3. When QUADIN = 2, FIRST 1 affects only the initial choice of α_i, and we see that this has far less effect.

Finally, consider the other extreme where interpolation is forced on all steps, even those in the QN-part. Once again, the initial choice of α_i does not have a marked effect. However, in contrast to rows to 8 to 10, when QUADIN = 3 the effect of the interpolation is now most pronounced for larger m. In fact, the set of function counts in each of the rows 12 to 14 is now much more constant. The decrease as m increases noted earlier is not so great. It is well-known that the BFGS algorithm performs best without exact searches, and indeed with α_i taken as 1 much of the time. Our results here confirm that VSCG has many of the characteristics of a quasi-Newton method, a property which becomes more pronounced as m increases. This is very much what we had hoped for.

The results of Section 5 were obtained with the parameter settings used in row 1. This seems the most appropriate for the strategy in the QN-part is like that of a quasi-Newton method, just as the CG-part is treated like a conjugate gradient code. The value of this choice is borne out by the results of Table 5.

7. Conclusions

We have described a new algorithm for minimization which combines the conjugate gradient and quasi-Newton methods. It is based on taking a particular view of the conjugate gradient method which we feel is very important. The algorithm is designed to have a variable storage requirement, and the numerical results which we have presented demonstrate quite clearly that the performance of the algorithm can be expected to be directly related to the amount of storage available. We have also given numerical results which illustrate that quasi-Newton and conjugate gradient algorithms require different strategies, and that the correct choice of strategy is vital if one is to have a successful algorithm.

Bibliography

[1] A. Buckley, "A combined conjugate gradient quasi-Newton minimization algorithm", *Mathematical Programming* 15 (1978) 200–210.

[2] A. Buckley, "Extending the relationship between the conjugate gradient and BFGS algorithms", *Mathematical Programming* 15 (1978) 343–348.

[3] A. Buckley, "Conjugate gradient methods", in: M.J.D. Powell, ed., *Nonlinear Optimization 1981, Proceedings of the NATO Advanced Research Institute on Nonlinear Optimization* (Academic Press, London, 1982) pp. 17–22.

[4] A. Buckley, "A portable package for testing minimization algorithms", in: John M. Mulvey, ed., *Proceedings of the COAL Conference on Mathematical Programming Software, Boulder, Colorado* (Springer, New York, 1982) 226–235.

[5] R. Fletcher and C.M. Reeves, "Function minimization by conjugate gradients", *Computer Journal* 7 (1963) 163–168.

[6] R. Fletcher, "A FORTRAN subroutine for minimization by the method of conjugate gradients", Report R7073, U.K. A.E.R.E., Harwell, England (1972).

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

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

[9] S.S. Oren and E. Spedicato, "Optimal conditioning of self-scaling variable metric algorithms", *Mathematical Programming* 10 (1976) 70–90.

[10] A. Perry, "A modified conjugate gradient algorithm", Discussion paper 229, Center for Mathematical Studies in Economics and Management Science, Northwestern University (1976).

[11] M.J.D. Powell, "Restart procedures for the conjugate gradient method", *Mathematical Programming* 12 (1977) 241–254.

[12] D.F. Shanno, "Conjugate gradient methods with inexact searches", *Mathematics of Operations Research* 3 (1978) 244–256.

[13] D.F. Shanno and K.-H. Phua, "Numerical comparison of several variable metric algorithms", *Journal of Optimization Theory and Applications* 25 (1978) 507–518.

[14] D.F. Shanno, "Remark on Algorithm 500", *ACM Transactions on Mathematical Software* 6 (1980) 618–622.

[15] Ph. Toint, "Some numerical results using a sparse matrix updating formula in unconstrained optimization", *Mathematics of Computation* 32 (1978) 839–851.