Mathematical Programming 15 (1978) 200–210.
North-Holland Publishing Company

A COMBINED CONJUGATE-GRADIENT QUASI-NEWTON MINIMIZATION ALGORITHM

A.G. BUCKLEY*

Concordia University, Montreal, Quebec, Canada

Received 29 November 1976
Revised manuscript received 11 April 1978

Although quasi-Newton algorithms generally converge in fewer iterations than conjugate gradient algorithms, they have the disadvantage of requiring substantially more storage. An algorithm will be described which uses an intermediate (and variable) amount of storage and which demonstrates convergence which is also intermediate, that is, generally better than that observed for conjugate gradient algorithms but not so good as in a quasi-Newton approach. The new algorithm uses a strategy of generating a form of conjugate gradient search direction for most iterations, but it periodically uses a quasi-Newton step to improve the convergence. Some theoretical background for a new algorithm has been presented in an earlier paper; here we examine properties of the new algorithm and its implementation. We also present the results of some computational experience.

Key words: Minimization, Optimization, Variable Metric, Conjugate-Gradient, Quasi-Newton.

1. Introduction

We will consider the problem of computing a point $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ which is a good approximation to a local minimum of a nonlinear function $f(\boldsymbol{x})$. It will be assumed that a subroutine is available for computing both $f(\boldsymbol{x})$ and the gradient vector $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{x})$, given $\boldsymbol{x}$. Second derivatives will not be used.

To solve a particular problem of this type, one commonly uses either a conjugate gradient (CG) algorithm or a variable metric or quasi-Newton (QN) algorithm. Each has its advantages. In general terms, a CG algorithm requires more iterations than a QN one to obtain an equally good $\boldsymbol{x}$, but on the other hand a CG algorithm requires little storage for implementation. Specifically, it does not require storage of a matrix as in a QN algorithm.

We will present an algorithm which combines the CG and QN methods in a manner based on the theory presented in Buckley [3]. We attempt to preserve the advantages of both kinds of algorithms. Thus, our algorithm will run without storage of any matrices but it will use more storage than the 3 or 4 vectors required by an ordinary CG algorithm. Its intended audience is therefore those people who have more than $4n$ locations available, but who still cannot handle the $O(n^2)$ locations required for QN methods. The actual amount of storage used

* This research was supported by the National Research Council of Canada grant number A-8962.

is variable and is determined by the availability of space, but as few as 8 vectors of length n will suffice.

In terms of the number of iterations required by the new algorithm, experience indicates what one would perhaps expect. Because more storage is available to hold information about the function, generally fewer iterations are used than in a CG algorithm, but a few more are required than in a QN algorithm.

For purposes of our description, it will be assumed that all line searches are exact. This point will be further elaborated upon in Section 5.

2. Background and motivation

The new algorithm, which we will dub 'CGQN', has been developed from four main observations.

First, QN updates consist of a sequence of rank 1 or rank 2 corrections to a matrix which normally begins as the identity. Therefore, if one limits the number of updates allowed and stores the vectors defining the rank 1 corrections explicitly, then far fewer than n^2 locations are needed to define and record the updated QN matrix. This is an idea which has appeared in Allwright [1] although he did not exploit it in the way which we are about to present. The QN updates to be considered are those rank 2 corrections which are members of the Broyden [4] class

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

where

$$U(x_k, H, \alpha) \equiv \left[I - \frac{\delta\gamma^T}{\delta^T\gamma}\right] H \left[I - \frac{\gamma\delta^T}{\delta^T\gamma}\right] + \frac{\delta\delta^T}{\delta^T\gamma} + \alpha w w^T,$$

$\delta \equiv \delta_k = x_k - x_{k-1}$ is the step taken,

$\gamma \equiv \gamma_k = g_k - g_{k-1}$ is the change in gradients,

$$w \equiv w_k = \frac{H\gamma}{\gamma^T H\gamma} - \frac{\delta}{\delta^T\gamma},$$

and where α is a scalar which determines the update formula.

Second, the conjugate gradient algorithm may be applied with an initial step which is not along the usual (see Fletcher [7]) steepest descent direction. In particular, let us follow the approach presented in several places (for example Allwright [1], Hestenes and Stiefel [8], Nazareth [10] and Powell [12]) and define a transformation of variables $y = H^{-1/2}x$, where H is a positive definite symmetric matrix. We may then apply the CG algorithm in the transformed coordinates, and we obtain a sequence of points and search directions satisfying the standard properties of the CG algorithm in those coordinates. This sequence of points and directions may be transformed back into the x coordinates, and what is important is to realize that we may obtain these same x coordinate points and

directions by applying a modification of the normal CG algorithm (call it TCG) directly in the x coordinates as follows:

Given x_0, define

$$d_1 = -Hg_0; \tag{2}$$

then, for $k = 1, 2, \ldots$, iterate:

$$x_k = x_{k-1} + \lambda_k d_k, \tag{3a}$$

$$\hat{\beta}_k = \frac{g_k^{\mathrm{T}} H(g_k - g_{k-1})}{g_{k-1}^{\mathrm{T}} H g_{k-1}}, \tag{3b}$$

$$d_{k+1} = -Hg_k + \hat{\beta}_k d_k. \tag{3c}$$

For the TCG algorithm, notice that the initial direction d_1 is not in the steepest descent direction. Naturally it reduces to the normal CG algorithm when $H = I$. And of course termination in at most n steps is still obtained for the TCG algorithm because it is equivalent to the normal y coordinate CG algorithm. Observe that we are following Powell [12] and choosing the Polak–Ribière form of CG algorithm.

Third, restarting is an essential part of CG minimization algorithms. This is discussed at some length in Powell [12 and 13]; however there are some comments we would like to include here. In considering a general smooth function $f(x)$, we know it is approximately quadratic near the minimum, so, in order to get good ultimate convergence, we must be able to solve a quadratic problem well. In particular, when f is quadratic, this normally means that we must have a steepest descent starting step $d_1 = -g_0$. However, in the general case, quadratic behavior is only local, so this starting step must be near the minimum. In other words, a restart strategy is demanded. Now a key point is that the TCG algorithm (3) also gives finite termination for a quadratic, so a restarting step of the form

$$d_{s+1} = -Hg_s \tag{4}$$

at the point x_s is quite acceptable, provided we continue with the algorithm (3) from that point on. Of course, if we restart with (4), we should certainly attempt to choose H meaningfully. If we have arrived at x_s with some matrix H defined, this suggests that we update H to H^* using a QN formula as in (1) and replace H by H^* before computing (4). If nothing else, this ensures that the so called QN formula

$$H\gamma = \delta$$

will hold at x_s.

Now, before we consider the fourth observation, we feel we should describe the new algorithm. The added notation should clarify the discussion of point four.

3. Description of the CGQN algorithm

The fundamental strategy we wish to present is the following. It is based on combining QN restarts of the form (4) with subsequent TCG steps.

From a given initial point x_0, begin the normal CG algorithm described by Polak [11]. As long as progress is satisfactory, continue the CG algorithm; what is "satisfactory" progress will be discussed in Section 5. When this is no longer the case, say at x_E, construct a QN update based on function information at x_{E-1} and x_E. This defines a matrix $H_1 = U(x_E, H_0, 0)$ which is a rank 2 update of $H_0 = I$ and which is stored by recording the values of the vectors $\delta_1 = x_E - x_{E-1}$ and $\eta_1 = H_0\gamma_1 = g_E - g_{E-1}$ which define the update.

Now suppose we are at a point x_E at which a positive definite matrix H_i has just been computed. We now rename x_E as x_s and restart the conjugate gradient algorithm. (Note that initially $x_s = x_0$ and $i = 0$.) Specifically, we define a QN search direction

$$d_{s+1} = -H_i g_s \tag{5}$$

and we do a line search along d_{s+1} to x_{s+1}. For further steps we generate TCG steps as defined by (3) with $H = H_i$.

Except when $x_s = x_0$, we note that, provided the line search along d_s is exact, eq. (3c) with $k = s$ in fact reduces to (5), for in this case it follows from (1) that $H_i(g_s - g_{s-1}) = H_i\gamma = \delta$ and so $\hat{\beta}_s = 0$. However we retain eq. (5) because in practice line searches will not be exact and at a restart we wish to be sure to begin with the QN direction d_{s+1} defined by (5) so that the properties of the subsequent TCG steps are not lost by a poor restarting step.

Using the directions (3c), these CG steps again continue until a point x_E is reached where it is decided that progress in reducing the function value is not satisfactory. Then a new matrix H_{i+1} is defined by storing a new update using $\delta_{i+1} = x_E - x_{E-1}$ and $\eta_{i+1} = H_i\gamma_{i+1} = H_i(g_E - g_{E-1})$. We then repeat from (5) with i replaced by $i + 1$ and with x_E again renamed x_s.

Notice that the QN updates are only done intermittently, so that the goal of reducing the storage needed for storing the QN update matrix has been achieved.

Before continuing, note the following regarding notation. The sequence of points generated by the algorithm will be denoted by $x_0, x_1, \ldots$. When a quasi-Newton update is computed, a subscript Q will be added. Thus we will have numbers $0 = Q_0 < Q_1 < Q_2 < \cdots$, and the matrix H_i will be computed when the

point x_{Q_i} is reached. In the notation above, x_s denotes a point x_{Q_i} and x_E denotes $x_{Q_{i+1}}$. The numbers $Q_1, Q_2, \ldots$ are not set in advance, but are determined dynamically, as explained in Section 5.

4. The fourth observation

Let us now examine the concept of intermittent updates more closely. For this discussion we will replace the local quadratic behavior of a general function f with a hypothetical example in which the function f is precisely quadratic in some neighborhood N of the minimum. Once a restart is done within N, at most n steps of the TCG algorithm will give the exact minimum.

In order to obtain finite termination once we enter N, a restart is needed, and preferably soon. Since we do not know when N is entered, frequent restarts are desirable. On the other hand, if we have restarted at $x_{Q_i} \in N$, and we restart frequently, there is a possibility that we will restart again at $x_{Q_{i+1}}$, this being before terminating. It is essential that we do not have to count n more steps from $x_{Q_{i+1}}$ before achieving termination, for then the same difficulty could arise again. In one case we know that a full n-count from $x_{Q_{i+1}}$ is not needed. If the algorithm of Section 3 is applied with a restart *every* step, we simply have a QN algorithm and it is well known that in at most n steps after entering N, we will terminate. Thus in at least one case, restarts of the form (5) with an updated H are worthwhile.

Now, we have already indicated that we only wish to update intermittently, so we must ask what effect this will have on termination. This is exactly the situation covered by the result in Buckley [3]. There it is shown that if we update intermittently at $x_{Q_1}, x_{Q_2}, \ldots$ (without loss of generality, all in N), and if we stick to the idea of updating H at each restart as in Section 3 (using (1) with $\alpha = 0$), then we do not affect our finite termination properties by doing unneeded restarts. That is, we terminate in at most n steps after the first restart in N at x_{Q_1}. It is this result which makes the use of intermittent updates sensible. The result in [3] also suggests only the use of the BFGS update, for otherwise, finite termination is affected by later restarts.

5. Implementation of the algorithm

In order to implement the algorithm described in Section 3, there are a number of important details which can not be overlooked. We will look at these now.

Positive definite H_i: It is clear from the derivation of the TCG algorithm that we require the matrices H_i to be positive definite. Now, it is well known that many QN updates generate a sequence of positive definite matrices H_i. In the present situation however, an update does not take place on every step and

therefore it must be verified that positive definite matrices are still obtained.

But this is straightforward, for proofs of positive definiteness of H^* in (1) depend only on the fact that H is positive definite and that $\boldsymbol{\delta}^T\boldsymbol{\gamma} > 0$. In particular then, providing that $\boldsymbol{\delta}^T\boldsymbol{\gamma} > 0$, as it is for an exact line search, the proof that H^* is positive definite is the same as that in Fletcher and Powell [5] when $\alpha = \boldsymbol{\gamma}^T H \boldsymbol{\gamma}$ (the DFP formula) and is the same as that in Fletcher [6] for other $\alpha \geq 0$ (this includes the BFGS formula). We will confirm shortly that in the practical case where the line search is not exact, we may still ensure that the condition $\boldsymbol{\delta}^T\boldsymbol{\gamma} > 0$ is obtained.

Downhill directions: Since H_i is positive definite, $\boldsymbol{d}_{s+1} = -H_i\boldsymbol{g}_s$ in (5) is downhill. Consider $k > s$. If the line search from $\boldsymbol{x}_{k-1}$ to $\boldsymbol{x}_k$ along $\boldsymbol{d}_k$ is exact, then from (3c),

$$\boldsymbol{d}_{k+1}^T\boldsymbol{g}_k = -\boldsymbol{g}_k^T H_i \boldsymbol{g}_k + \hat{\beta}_k \boldsymbol{d}_k^T \boldsymbol{g}_k = -\boldsymbol{g}_k^T H_i \boldsymbol{g}_k < 0 \tag{6}$$

and $\boldsymbol{d}_{k+1}$ is downhill. And of course it is imperative that the case of an inexact line search be considered, so let us do that now.

The line search: We require each line search to satisfy certain conditions of exactness. We have noted in (6) that the CG direction $\boldsymbol{d}_{k+1}$ given by (3c) is downhill provided that the line search leading to $\boldsymbol{x}_k$ is exact, but it is also clear from (6) that we will obtain a downhill direction whenever

$$(\boldsymbol{d}_k^T\boldsymbol{g}_k)\hat{\beta}_k = (\boldsymbol{d}_k^T\boldsymbol{g}_k)\frac{\boldsymbol{g}_k^T H_i(\boldsymbol{g}_k - \boldsymbol{g}_{k-1})}{\boldsymbol{g}_{k-1}^T H_i \boldsymbol{g}_{k-1}} < \boldsymbol{g}_k^T H_i \boldsymbol{g}_k. \tag{7}$$

In addition, when we do a QN update at $\boldsymbol{x}_k = \boldsymbol{x}_{Q_i}$, we insist that

$$\boldsymbol{\delta}^T\boldsymbol{\gamma} = (\boldsymbol{x}_k - \boldsymbol{x}_{k-1})^T(\boldsymbol{g}_k - \boldsymbol{g}_{k-1}) > 0$$

in order that H_i will be positive definite. This condition may clearly be replaced by the requirement that

$$\boldsymbol{d}_k^T\boldsymbol{g}_{k-1} < \boldsymbol{d}_k^T\boldsymbol{g}_k. \tag{8}$$

But we note that, although (8) is required only when an update takes place, we will not actually know if a QN update is required at $\boldsymbol{x}_k$ until after the line search is complete (as explained below). Thus we must insist that every line search satisfies both the conditions (7) and (8). Since these conditions are satisfied for an exact search, they may be attained in practice for any smooth function.

To conclude this section, we observe that we really require somewhat more, for it is generally accepted that one must take steps to ensure that each line search makes "reasonable" progress towards the one-dimensional minimum and that $\boldsymbol{d}_{k+1}$ is not nearly orthogonal to $\boldsymbol{g}_k$. One way of accomplishing this is to replace (7) and (8) with the stronger conditions

$$|\boldsymbol{d}_k^T\boldsymbol{g}_k| < K_1|\boldsymbol{d}_k^T\boldsymbol{g}_{k-1}|, \qquad (\boldsymbol{d}_k^T\boldsymbol{g}_k)\hat{\beta}_k < K_2(\boldsymbol{g}_k^T H_i \boldsymbol{g}_k)$$

where K_1 and K_2 are some small positive constants, say 0.2. These new conditions still hold for exact searches and hence are attainable.

The restart criterion: We now wish to ask: When should we update H_i? If the CG algorithm is applied to a quadratic, it is well known that successive gradients are orthogonal, i.e. $\boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{g}_j = 0$ for $j \neq t$. Using the algorithm given by (3) the corresponding result is that

$$\boldsymbol{g}_i^{\mathrm{T}} H \boldsymbol{g}_j = 0 \quad \text{for } j \neq t.$$

For a nonquadratic this relation certainly does not hold, but its deviation from 0 can be taken as an indication of how well the matrix $H = H_i$ is simulating the local quadratic behavior of f. This suggests comparing the value of $\boldsymbol{g}_i^{\mathrm{T}} H \boldsymbol{g}_j$ to 0 for certain t and j and restarting if the difference is deemed substantial.

In the test results of Section 6 we have used the test

$$\left| \frac{\boldsymbol{g}_{k-1}^{\mathrm{T}} H_i \boldsymbol{g}_k}{\boldsymbol{g}_k^{\mathrm{T}} H_i \boldsymbol{g}_k} \right| < \rho$$

to determine if a restart should be done at $\boldsymbol{x}_k$. Here ρ is a predefined constant, and the denominator is included to eliminate scaling effects. Tests indicate that performance of the algorithm is not particularly sensitive to the choice of ρ and any value such as 0.1 or 0.2 will do.

In a personal communication, Powell has suggested that in certain situations the test (9) may not be satisfactory because $\boldsymbol{g}_{k-1}^{\mathrm{T}} H_i \boldsymbol{g}_k$ could be 0 even when a restart is desirable. This has not occurred in tests done so far by the author, and restarts are in fact done reasonably frequently. (We recall from Section 4 that this is desirable and that there is no danger of restarting when we should not.) But nonetheless, in certain situations it may be desirable to find an alternative test for (9), and Powell suggests comparing the value

$$\boldsymbol{g}_{Q_i}^{\mathrm{T}} H_i \boldsymbol{g}_k$$

to 0 for $k > Q_i$, for in theory this is also 0 for a quadratic. We remark that this can be done only at the price of some increase in the storage required for the algorithm.

Storage of the updates: First observe that (1) can be written as

$$H^* = H + \boldsymbol{\eta}\left[\frac{\beta - 1}{b}\boldsymbol{\delta} - \frac{\beta}{a}\boldsymbol{\eta}\right]^{\mathrm{T}} + \boldsymbol{\delta}\left[\frac{b + (1-\beta)a}{b^2}\boldsymbol{\delta} - \frac{\beta - 1}{b}\boldsymbol{\eta}\right]^{\mathrm{T}} \tag{10}$$

where

$$a = \boldsymbol{\gamma}^{\mathrm{T}} H \boldsymbol{\gamma}, \qquad b = \boldsymbol{\delta}^{\mathrm{T}} \boldsymbol{\gamma}, \qquad \boldsymbol{\eta} = H\boldsymbol{\gamma}.$$

It is clear then that for each update we require $2n + 2$ locations in order to keep a, b, $\boldsymbol{\eta}$ and $\boldsymbol{\delta}$. (This takes into account the fact that we do not wish to recompute $H\boldsymbol{\gamma}$ each time it is needed.) If the number of updates is small this represents a substantial storage saving, as for example in the 5$^{\text{th}}$ problem of Table 1, Section 6, where we have $n = 60$. Here storage of H in matrix form would require 1830 locations for the more difficult to manage symmetric half, or 3600 in full. Now this example reached the minimum using only 14 updates, thus requiring a

comparable 1708 locations to store all of the updates. But what is important is that the new algorithm will still operate when this amount of storage is not available, and that it will not require a significant increase in the number of iterations needed for convergence. In the corresponding example of Table 2, we see that this is indeed the case. Here only 1 update is ever stored so only 122 locations are required, and a substantial improvement is still noted over the performance of the CG algorithm.

Computing with the factored form: To compare the computation required for the linear algebra when H_i is stored in factored form, we note the following. Suppose we are at x_k. A standard QN algorithm must update H_{k-1} to H_k and then compute $d_{k+1} = -H_k g_k$. For the commonly used BFGS update, this requires about $\frac{7}{2}n^2$ operations. In the new algorithm there are 2 cases: (a) where an update to H_i is done at x_k; (b) when a CG direction (3c) is used from x_k. In either case, the only significant computation per iteration is in finding $H_i g_k$ (providing $H_i g_{k-1}$ is stored, but that is required anyway for other reasons). In particular, no work (except $O(n)$) is required to update H_i to H_{i+1}, and even computing $H_{i+1} g_k$ in the event of an update is just an $O(n)$ operations modification to $H_i g_k$. Now, using (10), each term of $H_i g_k$ is formed by computing $\delta^T g_k$ and $\eta^T g_k$ and then adding scalar multiples of η and δ. For each update recorded this means $4n$ multiplies, so the i updates defining H_i use $4ni$ operations. Clearly then the new algorithm requires less work unless i is nearly n, and, with the philosophy behind this algorithm, that is most unlikely to occur.

Dropping update terms: Since one of the basic properties of this algorithm is that it will run in limited storage, one must decide what action to take when the allotted storage limit is reached. In particular, in order to store further QN updates, one must discard some of the old ones. There are several criteria which suggest themselves for picking the discards. However computational experience of the author has indicated none that is superior to the simple strategy of discarding all current updates and starting afresh.

One argument to support this choice concerns the matrices H_i. It is fundamental that they should be positive definite. When some of the rank one terms defining H_i are deleted several iterations after they were constructed, experience indicates that H_i almost always becomes non-positive definite. This may not however be immediately detected and can lead to nondownhill or poor search directions and numerical difficulties. These can be handled, but it seems best to simply avoid the problem.

6. Test results

We now give test results which indicate that the algorithm does display the behavior discussed earlier. The test problems are well known and can be found for example in Himmelblau [9], except for the 60 dimensional extended Powell

Table 1
Using unlimited updates

Function	n	Number of iterations	Number of function evaluations	Function value reached	Number of QN updates computed (CGQN algorithm)
Rosenbrock	2	31/22/24	103/64/65	$3_{-8}/9_{-9}/3_{-10}$	12
Helix	3	33/20/19	83/51/45	$2_{-12}/4_{-10}/3_{-10}$	10
Powell's Singular	4	35/18/16	86/39/35	$4_{-6}/1_{-7}/5_{-6}$	10
Woods	4	79/64/62	186/152/140	$8_{-12}/6_{-11}/4_{-9}$	27
Powell's Extended	60	55/21/15	130/49/35	$2_{-5}/3_{-7}/7_{-5}$	14

function given in Boland, Kamgnia and Kowalik [2]. In all cases termination was when $g^{\mathrm{T}}g \leq 10^{-6}$. In the tables, a_{-b} means $a \times 10^{-b}$.

First, in order to confirm that the idea of intermittent QN updates is indeed reasonable, we exhibit in Table 1 results comparing an ordinary CG strategy[1] (Polak–Ribière form), the mixed algorithm of Section 3 which does intermittent updates, and a standard QN update procedure[2] (BFGS update form). The figures in the table are in this order. For purposes of this test, no limit was placed on the storage available for the mixed algorithm. Clearly in most cases intermittent QN updates improve the convergence to nearly that of the standard QN procedure where an update is made every step. Although we note that in one case the mixed algorithm is even better than QN, all we wish to conclude is that the intermittent updates do tend to significantly improve the performance of an ordinary CG algorithm.

Of course the object of the new algorithm is to operate in limited storage and so in Table 2 we give the figures obtained by limiting the storage available to the new algorithm and repeating the computations of Table 1 for the mixed algorithm (these are the first of each pair of figures). Comparing the leading entries in Table 2 to those in Table 1 we see that generally little is lost. A substantial improvement over the CG algorithm is still obtained. Note that the final example is included to illustrate the behavior of the algorithm on a medium sized problem, which is of course where we hope this algorithm to be beneficial.

In each entry, the second of the pair of figures provides a comparison to the routine VA14A. This code is from the Harwell library and it implements a

[1] These figures were obtained using an implementation of the algorithm CGQN. By setting $\rho = \infty$, no updates are ever done. The normal steepest descent restart was done every n iterations. These figures are entirely comparable to those of well-known CG implementations such as VA08 in the Harwell library.

[2] These figures were obtained by setting $\rho = -1$, thus forcing a QN step on every iteration; they are entirely comparable, for example, to Fletcher's VA13 variable metric algorithm in the Harwell library.

Table 2
CGQN using limited storage vs. VA14A

Function	n	Number of iterations	Number of function evaluations	Function value reached	Number of QN updates computed (CGQN algorithm)	Number of updates stored (CGQN algorithm)
Rosenbrock	2	22/23	72/62	$3_{-9}/2_{-8}$	15	1
Helix	3	21/23	49/56	$4_{-9}/3_{-10}$	12	1
Powell's Singular	4	24/21	57/49	$3_{-6}/9_{-7}$	11	1
Woods	4	74/a	178/a	1_{-9}/a	32	1
Powell's Extended	60	28/27	68/67	$1_{-6}/3_{-7}$	12	1

[a] VA14A failed to reach the minimum in this case.

modified conjugate gradient algorithm with restarts which is due to Powell [12]. Powell has indicated that his algorithm performs substantially better than ordinary CG implementations; here we see that the proposed CGQN algorithm is comparable in its performance with VA14A.

7. Conclusion

We have designed an algorithm which combines a CG algorithm with intermittent QN updates and we have demonstrated that this idea does indeed lead to improved convergence over the standard CG algorithm, even when only very few (e.g. 1 or 2) updates may be stored at any one time.

Acknowledgment

The author would like to thank Professor M.J.D. Powell for his careful reading of drafts of this paper and for his perceptive comments and helpful suggestions.

References

[1] J.C. Allwright, "Improving the conditioning of optimal control problems using simple models", in: D.J. Bell, ed., *Recent mathematical developments in control* (Academic Press, London, 1972).

[2] W.R. Boland, E. Kamgnia and J.S. Kowalik, "A conjugate gradient optimization method invariant to nonlinear scaling", Report TR 245, Department of Mathematical Sciences, Clemson University, Clemson, SC (1977).

[3] A.G. Buckley, "Extending the relationship between the conjugate gradient and BFGS algorithms", *Mathematical Programming*, to appear.
[4] C.G. Broyden, "The convergence of a class of double rank algorithms, Part I", *Journal of the Institute of Mathematics and its Applications* 7 (1971) 76–90.
[5] R. Fletcher and M.J.D. Powell, "A rapidly convergent descent method for minimization", *The Computer Journal* 7 (1963).
[6] R. Fletcher, "A new approach to variable metric algorithms", *The Computer Journal* 13 (1970) 317–322.
[7] R. Fletcher, "Conjugate direction methods", in: W. Murray, ed., *Numerical methods for unconstrained optimization* (Academic Press, London, 1972) pp. 73–86.
[8] M.R. Hestenes and E.L. Stiefel, "Methods of conjugate gradients for solving linear systems", *Journal of Research of the National Bureau of Standards* 49 (1952) 409–436.
[9] D.M. Himmelblau, *Applied nonlinear programming* (McGraw-Hill, New York, 1972).
[10] L. Nazareth, "A relationship between the BFGS and conjugate gradient algorithms", AMD Tech. Memo 282, Argonne National Laboratory (1977).
[11] E. Polak, *Computational methods in optimization: A unified approach* (Academic Press, New York, 1971).
[12] M.J.D. Powell, "Restart procedures for the conjugate gradient method", *Mathematical Programming* 12 (1977) 241–254.
[13] M.J.D. Powell, "Some convergence properties of the conjugate gradient method", *Mathematical Programming* 11 (1976) 42–49.