J. Austral. Math. Soc. Ser. B 31(1989), 122–134

A CONJUGATE DIRECTION IMPLEMENTATION OF THE BFGS ALGORITHM WITH AUTOMATIC SCALING

IAN D. COOPE[1]

(Received 20 March 1988; revised 28 August 1988)

Abstract

A new implementation of the BFGS algorithm for unconstrained optimisation is reported which utilises a conjugate factorisation of the approximating Hessian matrix. The implementation is especially useful when gradient information is estimated by finite difference formulae and it is well suited to machines which are able to exploit parallel processing.

1. Introduction

Quasi-Newton methods for the unconstrained minimisation of $f(x)$, $x \in \mathbf{R}^n$, are line search algorithms which use the basic iteration

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}, \qquad k = 1, 2, \ldots, \tag{1.1}$$

to generate a sequence of approximations $\{x^{(k)}, k = 2, 3, \ldots\}$ to a stationary point, x^*, of $f(x)$ from a given starting vector $x^{(1)}$.

Convergence of the iterative scheme (1.1) is normally achieved by choosing the scalar $\alpha^{(k)} > 0$ to reduce the objective function, $f(x)$, at each iteration by satisfying a descent condition of the form

$$f(x^{(k+1)}) < f(x^{(k)}) + \rho \alpha^{(k)} p^{(k)\mathsf{T}} \nabla f(x^{(k)}), \qquad \rho \in (0, 1/2), \tag{1.2}$$

which is a little stronger than requiring $f(x^{(k+1)}) < f(x^{(k)})$.

The search direction, $p^{(k)} \in \mathbf{R}^n$, in equation (1.1) is determined by solving a linear system of equations

$$B^{(k)} p = -g^{(k)}, \tag{1.3}$$

[1]Department of Mathematics, University of Canterbury, Christchurch, N.Z.

where $B^{(k)}$ is a positive definite approximation to the $n \times n$ Hessian matrix of second derivatives, $\nabla^2 f(x^{(k)})$, and $g^{(k)} \in \mathbf{R}^n$ denotes the gradient vector $\nabla f(x^{(k)})$. Often the choice $B^{(1)} = I$ is made and we consider the case when the highly successful BFGS formula (Broyden [2], Fletcher [4], Goldfarb [6], Shanno [14]),

$$B^{(k+1)} = \left[B - \frac{Bpp^{\mathsf{T}}B}{p^{\mathsf{T}}Bp} + \frac{\gamma\gamma^{\mathsf{T}}}{\alpha p^{\mathsf{T}}\gamma}\right]^{(k)}, \tag{1.4}$$

where $\gamma^{(k)}$ is the vector

$$\gamma^{(k)} = g^{(k+1)} - g^{(k)}, \tag{1.5}$$

is used to update the matrix $B^{(k)} \in \mathbf{R}^{n\times n}$.

Instead of working directly with the matrix B, or its inverse, most modern implementations store and update the Choleski factors of B since this enables the search direction p to be obtained in $O(n^2)$ floating point operations in a numerically stable manner. Recently, however, Han [8], and Powell [11] have considered updating factorisations of the inverse to the approximating Hessian matrix. In order to describe their approach we introduce some extra notation.

If S is a nonsingular $n \times n$ matrix satisfying

$$B^{-1} = SS^{\mathsf{T}}, \tag{1.6}$$

then S is defined to be a *conjugate factorisation* of the matrix B, and we note that the columns of S, $\{s_i, i = 1, 2, \ldots, n\}$ satisfy the conjugacy conditions $s_i^{\mathsf{T}}Bs_j = 0$, $i \neq j$. Clearly S is not defined uniquely by (1.6) because S may be post-multiplied by any $n \times n$ orthogonal matrix without changing the definition of B.

Now, the search vector, $p = -B^{-1}g$, can be calculated in $2n^2$ operations by first forming the vector

$$y = S^{\mathsf{T}}g, \tag{1.7}$$

then $p = -Sy$. The vector $y \in \mathbf{R}^n$ has components

$$y_i = s_i^{\mathsf{T}}\nabla f, \qquad i = 1, 2, \ldots, n, \tag{1.8}$$

which are the directional derivatives of $f(x)$ in the directions $\{s_i\}_1^n$. This latter observation is important if gradient information is estimated by finite difference formulae because it allows the possibility of calculating approximations to the vector y directly. Thus if forward differences are used then

$$y_i = (f(x + h_i s_i) - f(x))/h_i + O(h_i). \tag{1.9}$$

If, however, the more accurate central difference formula,

$$y_i = (f(x + h_i s_i) - f(x - h_i s_i))/(2h_i) + O(h_i^2), \tag{1.10}$$

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

is used, then there is an added bonus in that some second derivative terms can be estimated without requiring any extra function evaluations since

$$s_i^T[\nabla^2 f(x)]s_i = (f(x + h_i s_i) - 2f(x) + f(x - h_i s_i))/h_i^2 + O(h_i^2). \quad (1.11)$$

This estimated second derivative may be used immediately to rescale the length of the vector s_i, and we refer to this process as *automatic scaling.* Note that a different differencing interval h_i, $i = 1, 2, \ldots, n$, has been used for each of the directions $s_i, i = 1, 2, \ldots, n$, in (1.9)–(1.11) since the lengths of these vectors may be quite different.

In the next section we show how to rewrite the BFGS formula in terms of the conjugate factorisation (1.6) in a way that allows automatic scaling to be incorporated easily. Section 3 describes a particular implementation with information on line searches, choices of finite difference intervals and precise details of how automatic scaling is applied. Section 4 gives numerical results for a FORTRAN program that has been successfully applied to several test problems and the final section discusses some possible difficulties and further developments.

2. The BFGS formula with automatic scaling

Brodlie, Gourlay and Greenstadt [1] show that (1.4) can be written in product form and this is used by Han [8] to give an equivalent form of the BFGS formula applied to update the conjugate factorisation of $B^{(k)}$ directly as

$$S^+ = [I - pq^T]S, \quad (2.1)$$

where q is the vector

$$q = \gamma/(p^T\gamma) + g/(-p^T g p^T \gamma/\alpha)^{1/2}, \quad (2.2)$$

and where we now use the superscript + to denote items evaluated at iteration $(k + 1)$ and drop superscripts for items evaluated at iteration (k).

Equations (1.6), and (2.1) show that the BFGS algorithm is easily implemented without storing the matrix B if, instead, the matrix S is stored and updated. Han [8] favours the use of conjugate factorisations because it allows the exploitation of parallel processors in a natural way. However, even on serial machines, the results reported in Section 4 show that considerable improvements in efficiency are possible. For reasons of numerical stability, Powell [1] prefers first to post-multiply S by an appropriate orthogonal matrix and his approach is easily incorporated in the scheme that we have in mind. This is particularly important when $\|S^+\| \ll \|S\|$ but we keep the notation simpler by not considering this aspect.

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

Interestingly, we note that the form of the BFGS formula suggested by (2.1) is not new since it was used by Davidon [3] and by Osborne and Saunders [9] more than ten years earlier than the work described in [8], [11]. This form is still not suitable for our use because the vectors y and g are not directly available. Therefore, we rewrite the updating formula using (1.3), (1.6) and (1.7) to replace references to gradient vectors by directional derivatives. The resulting formula is

$$S^+ = S + pv^\mathsf{T}, \tag{2.3}$$

where the vector v depends on the directional derivative vectors $y = S^\mathsf{T}g$ and $\bar{y} = S^\mathsf{T}g^+$ through the simple equation

$$v = z/(y^\mathsf{T}z) - y/(-y^\mathsf{T}yy^\mathsf{T}z/\alpha)^{1/2}, \tag{2.4}$$

where z is the vector difference

$$z = \bar{y} - y. \tag{2.5}$$

Finally we require the vector $y^+ = [S^+]^\mathsf{T}g^+$ for the start of the next iteration. This is obtained by updating the vector $\bar{y}$ to reflect the change of basis directions using (1.7) and (2.3) as

$$y^+ = \bar{y} - (y^\mathsf{T}\bar{y})v. \tag{2.6}$$

After applying the updating formulae (2.3) and (2.6) we now have all the information required for the next iteration starting with the calculation of the new search direction $p^+ = -S^+y^+$.

This form of the BFGS algorithm allows automatic scaling to be implemented easily. We simply rescale the columns of S using (1.11) to make the current conjugate factorisation agree with the estimated second derivative terms whenever the central difference formula (1.10) is used to estimate a first directional derivative. This rescaling of the columns of S will also necessitate an adjustment in the first directional derivatives y and $\bar{y}$ but this is a trivial change which is described in the next section. Once the rescaling is performed the BFGS formula is applied exactly as before using (2.3)–(2.6).

Of course rescaling can also be incorporated when analytic gradients are used but this requires extra function or gradient evaluations and is, therefore, less attractive since it may be argued that these extra evaluations would be better spent in a more direct attempt at reducing f. It may still be useful, however, in circumstances where speed is important and a parallel processing

machine is available because automatic scaling can significantly reduce the number of iterations required.

3. A conjugate direction implementation

We consider in reasonable detail a computational algorithm implementing the ideas outlined in the previous section. The algorithm uses function values only, employing the finite difference formula (1.9) or (1.10) to estimate first directional derivatives when required. First we summarise the basic algorithm. We assume that S, x and y are available at the start of each iteration. For the first iteration these values must be given or estimated. For example, $S = I$, the identity matrix, x an initial guess and $y = S^T \nabla f(x)$, or a calculated estimate based on (1.9) or (1.10) are suitable values. Then the following steps complete an iteration.

(1) Calculate the search direction: $p = -Sy$.
(2) Determine α satisfying: $f(x + \alpha p) < f(x)$.
(3) Form the new estimate: $x^+ = x + \alpha p$.
(4) Estimate the vector $\bar{y}$ of directional derivatives at x^+ with respect to the current basis directions S using either (1.9) or (1.10).
(5) If (1.10) was used in step (4) rescale S, y and $\bar{y}$ using (1.11) to estimate second directional derivatives at x^+.
(6) If $y^T y > y^T \bar{y}$ calculate S^+ and y^+ from (2.3)–(2.6). Otherwise, set $S^+ = S$, $y^+ = \bar{y}$.

It should be clear that the basic algorithm cannot be implemented until the line search procedure (step (2)) is defined and the procedures for estimating directional derivatives and rescaling are made precise (steps (4) and (5) above). Strictly, we should also consider a terminating criterion but we take the view that a terminating criterion should only influence the number of points an algorithm generates and not the essential nature of the points. Although this view may not be universally upheld it is, nevertheless, a simple matter to incorporate different stopping rules and so we do not consider this aspect in detail here. (This is not to say that the terminating rule is unimportant!)

3.1 Derivative estimation.

A simple approach would be to use the central difference formula (1.10) at every iteration. This has the advantage of also allowing automatic scaling to be applied at every iteration. Preliminary tests indicated that this approximately halved the number of iterations required to reduce the objective function to below a prespecified threshold value. The disadvantage is

that twice as many function evaluations are required compared to the use of (1.9) which does not permit automatic scaling. Thus if only low accuracy is required then there is little to be gained by using the central formula exclusively. If high accuracy is required then central differences must always be used in the later iterations and then the extra second derivative information provided by (1.10) can profitably be incorporated in the matrix S.

The decision on when to switch from forward to central differences is a delicate one; switching too soon is clearly inefficient, but delaying too long results in gross errors due to cancellation. These errors are inherited by S and it may take many iterations to recover a good conjugate factorisation matrix. For the results reported in Section 4, this decision was based on the relative size of the differencing interval h_i, $i = 1, \ldots, n$, compared to the components of the displacement vector $x^+ - x = -\alpha S y$. Specifically, the central difference formula (1.10) was used to estimate $\bar{y}_i$ if

$$|\alpha y_i| < 10 h_i. \tag{3.1}$$

It was clear from initial tests, however, that automatic scaling was also desirable in the early iterations so central differences were used every fourth iteration even if the test (3.1) failed. Thus for the results presented in Table 4.2 automatic scaling was applied at least every fourth iteration and ultimately on every iteration as the directional derivatives $\{y_i\}$ tend to zero. The default values for the differencing intervals were set to

$$h_i = 10^{-6}/\|s_i\|, \qquad i = 1, \ldots, n, \tag{3.2}$$

although there is provision to adjust these values in the FORTRAN program. These values were also used in the Harwell library routing VA10AD, (Fletcher's [4] implementation of a finite difference, quasi-Newton algorithm), which was used as a basis for comparison on some problems. It should be noted that the efficiency of optimisation algorithms which make use of finite difference estimates of derivatives can be affected seriously by inappropriate choice of differencing intervals. The choice (3.2) was made after preliminary testing of values of h_i in the range $[\varepsilon^{1/4}, \varepsilon^{1/2}]/\|s_i\|$ where ε denotes the machine precision (approximately 10^{-16} for the machines used). Of course the values (3.2) should not be used if the current value of $\|x\|$ is very large or very small. Special precautions were taken in such cases to ensure that $\varepsilon^{1/4}\|x\| \leq h_i\|s_i\| \leq \varepsilon^{1/2}\|x\|$. Details can be found in the FORTRAN program. Finally we note that a good discussion on choosing finite difference intervals is given in Gill et al [5].

3.2 Automatic scaling.

Post-multiplying S by a diagonal matrix, $D = diag[d_1, \ldots, d_n]$, scales the columns of S and this affects the values of the directional derivatives y and

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

$\bar{y}$ by the same scaling factors. Therefore, automatic scaling is achieved by simply replacing S, y and $\bar{y}$ by SD, Dy and $D\bar{y}$ for a suitably chosen diagonal matrix D. The values used for d_i, $i = 1, \ldots, n$, are usually

$$d_i = h_i/[f(x^+ + h_i s_i) - 2f(x^+) + f(x^+ - h_i s_i)]^{1/2}, \tag{3.3}$$

but if the second derivative estimate is negative (and hence d_i complex) or if the computed value of (3.3) is greater than $\sqrt{10}$ then we set $d_i = \sqrt{10}$ to prevent the component of the search direction due to s_i from becoming too large too quickly. Thus if negative curvature is detected, the effect is to increase the components of the search vector in the directions of negative curvature by a factor of 10 at each iteration until either positive curvature is encountered or the objective function is deemed to be unbounded below. The factor 10 was chosen arbitrarily. This strategy is easily implemented but we note that many other possibilities exist. In particular, directions of negative curvature could be searched as soon as discovered until positive curvature estimates are retrieved.

3.3 The line search.

In this implementation a very simple line search was used. At each iteration an attempt is made to satisfy (1.2) with $\rho = .1$, except that available estimated derivative information is used to replace the gradient value. Specifically, if $\alpha = 1$ satisfies (1.2) then the line search terminates immediately. Otherwise α is replaced by $\max(.1, \beta)$ where β minimises the quadratic function which interpolates the values $f(x)$ and $f(x+\alpha p)$ and whose derivative at $\alpha = 0$ agrees with the estimated derivative information. Thus a new value of α is obtained and the test for acceptability reapplied. This process is repeated at most ten times, after which the value of α yielding the smallest function value is accepted. If the accepted value is zero then the algorithm terminates with a diagnostic advising the user that accuracy is limited through the machine precision or through the choice of inappropriate finite differencing intervals. Usually an acceptable value for α is obtained in one or two trials.

3.4 Maintaining positive definiteness.

A well-known property of the BFGS formula (1.4) is that the matrix $B^{(k+1)}$ is positive definite provided that $B^{(k)}$ is positive definite and

$$(p^T y)^{(k)} > 0. \tag{3.4}$$

Thus, when gradient information is available, positive definiteness is easily maintained by imposing (3.4) in addition to (1.2) in the test for acceptability of $\alpha^{(k)}$ in the line search. In the present context, this condition can be shown to be equivalent to requiring $y^T z < 0$, or equivalently,

$$y^T y > y^T \bar{y}. \tag{3.5}$$

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

Of course, conditions (1.2) and (3.5) may not be achievable when rounding errors are present and y is estimated by finite difference formulae. Therefore, we prefer to keep the simple form of the line search described in Section 3.3 and choose not to apply the BFGS update if (3.5) is not satisfied. This decision is reflected in step (6) of the algorithm summary above. Again, there are alternative strategies (see for example Powell [10]) that could be adopted here but we note that even if the BFGS formula is not used because of negative curvature information, automatic scaling may still be applied to give an improvement in the conjugate factorisation matrix.

4. Numerical results

The algorithm described in the previous sections has been programmed in FORTRAN77 and is available in the Harwell subroutine library under the name VF04AD. Preliminary results indicate that it is a significant improvement on more standard implementations that do not incorporate automatic scaling but caution must be exercised when making comparisons between different algorithms. Because we were interested in establishing that the technique of automatic scaling was in itself a valuable addition to an optimisation algorithm, it was important to rule out differences due to other aspects of the implementation, for example differences in line searches and differences due to the choice of finite difference intervals and the representation of second derivative information. Therefore, at each stage in the development of the new algorithm a comparison was made with the Harwell library routine VA13AD [12] which is a gradient version of a quasi-Newton algorithm employing the BFGS formula to update the Choleski factors of the approximating Hessian matrix. VA13AD was gradually transformed into VF04AD by a sequence of small changes which allowed a direct comparison between the two algorithms at each stage. Thus, although the final version of VF04AD had very little in common with VA13AD it was possible to identify precisely which of the changes contributed most to improved performance. The test program used for validation of VA13AD for the Harwell library was also used extensively in the development of VF04AD. This program contains a subroutine defining a function of 55 variables, (see Appendix), which is large enough to demonstrate the effects of automatic scaling quite conclusively; we refer to this function as F55.

The first stage was to implement the conjugate factorisation of B without automatic scaling instead of the Choleski factors of B while keeping all other aspects of the two algorithms identical. This resulted in virtually no change in the results for problem F55; VA13AD and the modified code each required

70 iterations to achieve 14 significant figures in the optimal value of the objective function (note that this is close to the limit of the machine accuracy in double precision). In exact arithmetic the results would, of course, be indistinguishable.

The next stage gave more interesting results because automatic scaling was introduced at every iteration in the modified code as described in Section 3.2 above. Analytic gradients were still used in the calculation of the search direction and the line search algorithm of VA13AD was unchanged. The effect of automatic scaling on VA13AD is considerable; the results of Table 4.1 show that more than twice as many iterations are required without automatic scaling for full accuracy in $f(x^*)$.

TABLE 4.1

Iterations of VA13AD on F55		Accuracy
no scaling	with scaling	$f(x) \leq$
0	0	104.1214111280980
10	12	1.197264928395396
20	16	0.160549957009795
30	21	0.132662338268089
40	22	0.132480234631059
50	26	0.132470104997354
60	28	0.132470103795027
70	30	0.132470103792991
76	31	0.132470103792989

The entries in column three of Table 4.1 represent the actual value of $f(x)$ attained by the unmodified version of VA13AD after the number of iterations indicated in column one. The entries in column two represent the number of iterations to give at least the same accuracy with the modified algorithm which incorporates automatic scaling at every iteration. Notice, however, that automatic scaling has not given an improvement in the first ten iterations for this problem. The highly nonlinear terms of this function cause the second derivative information to change rapidly in the first few iterations and there is little difference in the two algorithms in the early stages. As soon as the objective function can be adequately modelled by a quadratic the improvement in the modified algorithm is quite striking. It is unusual to see a quasi-Newton algorithm achieve such high accuracy in less than n

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

iterations but it should be remembered that, with automatic scaling, twice as much information is being used to modify the estimated second derivative information at each iteration. Of course, $2n$ extra function evaluations must be made at every iteration in order to estimate the second derivative information and we have already pointed out that this is wasteful unless these evaluations are necessarily made when estimating the first directional derivatives using (1.10). Therefore, the final stage in the construction of the new algorithm was to implement the remaining features as described in Section 3. The results on some standard (and well known) test functions are presented in Table 4.2 which gives the number of iterations and evaluations of $f(x)$ required to obtain an accuracy $f(x) - f(x^*) < 10^{-14}$.

The Harwell library routine VA10AD was also applied to the same test problems and results are included in parentheses in Table 4.2. VA10AD is also a finite difference quasi-Newton algorithm but second derivative information is represented by Choleski factors and sometimes the DFP updating scheme is used instead of the BFGS scheme [4]. This algorithm uses a more accurate line search than VF04AD and uses a different strategy to choose between forward and central difference formulae.

TABLE 4.2

Results for VF04AD (VA10AD)			
Problem	n	Iterations	Evaluations
Rosenbrock	2	25 (36)	142 (177)
Helix	3	27 (28)	146 (164)
Hilbert	5	13 (20)	264 (220)
Wood	4	73 (50)	548 (347)
Powell	4	34 (37)	249 (313)
F55	55	23 (51)	1868 (4139)

The results for Table 4.1 were obtained on the IBM 3084Q mainframe computer at Harwell and those in Table 4.2 were obtained on a SUN 3/160 micro-computer system at the author's own institution.

5. Discussion

The algorithm implementation described in this paper could be further enhanced if a true parallel processing machine (MIMD machine, see [13]

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

for example) were available. Each processor could be directed to the task of determining a finite difference estimate along each of the directions s_i, $i = 1, \ldots, n$, in parallel; indeed line searches could be performed in parallel or the conjugate subspace approach described by Han [8] could easily be adopted. Clearly there is much scope for further investigation here. However, the results of the previous section show that even for the simplest of implementations the conjugate direction approach can give considerable advantages even on serial machines.

The work described here could also be extended to handle constrained minimisation problems. Suitable approaches to the linearly constrained problem have been discussed by Davidon [3] and by Powell [11]. However, the conjugate factorisation approach can also be adapted to handle nonlinear constraints. Indeed subroutine VF04AD has already been extended to handle nonlinear constraints using simple penalty function ideas in an algorithm designed to be easy to use. We do not consider the details here because the author believes that it would be more efficient to make use of sequential quadratic programming techniques. The quadratic programming algorithm of Goldfarb and Idnani [7] is particularly attractive here because it requires a conjugate factorisation of the appropriate Hessian matrix. These ideas are currently being investigated by the author and will be reported at a later date.

Acknowledgement

This research was supported by the U. K. Atomic Energy Research Establishment.

Appendix
Test program used to validate VA13AD

```
      DOUBLE PRECISION F,G,SCALE,W,X,XD,YD
      COMMON/VA13BD/IPRINT,LP,MAXFUN,MODE,NFUN
      COMMON/XXX/XD,YD
      DIMENSION XD(51),YD(51),X(55),G(55),SCALE(55),W(1870)
      EXTERNAL F55
      DO 1 I=1,51
      SCALE(I)=0.1D0
      XD(I)=0.125664D0*DBLE(I-1)
      YD(I)=SIN(XD(I))
    1 X(I)=(1.0D0+0.5D0*YD(I))*XD(I)
```

```
      DO 2 I=52,55
      SCALE(I)=0.1D0
2     X(I)=0.0D0
      IPRINT=1
      MAXFUN=100
      ACC=1.0D-14
      CALL VA13AD(F55,55,X,F,G,SCALE,ACC,W)
      STOP
      END
      SUBROUTINE F55(N,X,F,G)
      DOUBLE PRECISION C,F,G,X,XD,YD
      COMMON/XXX/XD,YD
      DIMENSION XD(51),YD(51),X(55),G(55)
      F=0.0D0
      DO 1 I=52,55
1     G(I)=0.0D0
      DO 2 I=1,51
      C=X(52)+X(I)*(X(53)+X(I)*(X(54)+X(I)*X(55)))-YD(I)
      F=F+C*C+(X(I)-XD(I))**2
      G(I)=2.0D0*((X(53)+X(I)*(2.0D0*X(54)+3.0D0*X(I)*X(55)))*C+X(I)-XD(I))
      DO 2 J=52,55
      G(J)=G(J)+2.0D0*C
2     C=C*X(I)
      RETURN
      END
```

References

[1] K. W. Brodlie, A. R. Gourlay and J. Greenstadt, "Rank-one and rank-two corrections to positive definite matrices expressed in product form", *J. Inst. Maths. Applics.* **11** (1973) 73–82.

[2] C. G. Broyden, "The convergence of a class of double rank minimization algorithms. 2. The new algorithm", *J. Inst. Maths. Applics.* **6** (1970) 222–231.

[3] W. C. Davidon, "Optimally conditioned optimization algorithms without line searches", *Math. Prog.* **9** (1975) 1–30.

[4] R. Fletcher, "A new approach to variable metric algorithms", *Computer Journal* **13** (1970) 317–322.

[5] P. E. Gill, W. Murray, M. H. Wright, *Practical Optimization* (Academic Press, London, 1982).

[6] D. Goldfarb, "A family of variable metric methods derived by variational means", *Maths of Comp.* **24** (1970) 23–26.

https://doi.org/10.1017/S0334270000006524 Published online by Cambridge University Press

[7] D. Goldfarb and A. Idnani, "A numerical stable dual method for solving strictly convex quadratic programs", *Math. Prog.* **27** (1983) 1–33.
[8] S-P. Han, "Optimization by updated conjugate subspaces", in *Numerical Analysis* (eds. D. F. Griffiths and G. A. Waton), (Pitman Research Notes in Mathematics Series 140, Longman Scientific & Technical, Burnt Mill, England, 1986) 82–97.
[9] M. R. Osborne and M. A. Saunders, "Descent methods for minimization", in *Optimization* (eds. R. S. Anderssen, L. S. Jennings, D. M. Ryan), (University of Queensland Press, St. Lucia, 1972) 221–237.
[10] M. J. D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations", in *Numerical Analysis, Lecture Notes in Mathematics* 630 (ed. G. A. Watson), (Springer-Verlag, 1978) 144–157.
[11] M. J. D. Powell, "Updating conjugate directions by the BFGS formula", *Math. Prog.* **38** (1987) 29–46.
[12] M. J. D. Powell, "Subroutine VA13AD", *Harwell Subroutine Library* (1975) U. K. Atomic Energy Research Establishment.
[13] U. Schendel, *Introduction to numerical methods for parallel computers* (Ellis Horwood, Chichester, 1984).
[14] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization", *Maths of Comp.* **24** (1970) 647–656.