PERFORMANCE OF VARIOUS BFGS IMPLEMENTATIONS WITH LIMITED PRECISION SECOND-ORDER INFORMATION

D. BYATT[1], I. D. COOPE[1] and C. J. PRICE[1]

Abstract

The BFGS formula is arguably the most well known and widely used update method for quasi-Newton algorithms. Some authors have claimed that updating approximate Hessian information via the BFGS formula with a Cholesky factorisation offers greater numerical stability than the more straightforward approach of performing the update directly. Other authors have claimed that no such advantage exists and that any such improvement is probably due to early implementations of the DFP formula in conjunction with low accuracy line searches.

This paper supports the claim that there is no discernible advantage in choosing factorised implementations (over non-factorised implementations) of BFGS methods when approximate Hessian information is available to full machine precision. However the results presented in this paper show that a factorisation strategy has clear advantages when approximate Hessian information is available only to limited precision. These results show that a conjugate directions factorisation outperforms the other methods considered in this paper (including Cholesky factorisation).

1. Introduction

Quasi-Newton algorithms are used to solve the local optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

iteratively, where $f : \mathbb{R}^n \to \mathbb{R}$ and gradient information is available. The solution is attained when $\nabla f(x) = 0$, but in practice the usual requirement is that $\|\nabla f(x)\| \leqslant \tau_g$ for some (typically small) positive constant τ_g.

Throughout this paper the convention of writing $f(x_k)$ as f_k and $\nabla f(x_k)$ as g_k is used. At iteration k of a quasi-Newton method a search direction p_k is found by

---

[1]Department of Mathematics and Statistics, University of Canterbury, Private Bag 4800, Christchurch, New Zealand; e-mail: d.byatt@math.canterbury.ac.nz.

solving the system of equations

$$B_k p_k = -g_k \tag{1.1}$$

where B_k approximates, in some sense, the Hessian matrix $\nabla^2 f(x_k)$. A line search is then performed along $x_k + \alpha p_k$, $\alpha \in \mathbb{R}$ to find a new iterate $x_{k+1} = x_k + \alpha_k p_k$ for some α_k that satisfies the line search criteria. Information at this new point is used to generate a new approximate Hessian matrix B_{k+1}. If B_k is positive definite then $p_k^\mathsf{T} g_k < 0$ so that p_k is a descent direction for f.

The use of Cholesky factors of the approximate Hessian matrices B_k was introduced in [7] and is now in widespread use. Proponents of this implementation claim it avoids the computational instability of using the inverses of the approximate Hessian matrices and allows the efficient calculation of the search direction in $O(n^2)$ operations by using both forward and back substitution. The standard Cholesky factorisation implementation of the BFGS method uses the *modified* Cholesky factorisation $B_k = L_k D_k L_k^\mathsf{T}$ where L_k is unit lower triangular and D_k is diagonal. The modified implementation allows the easy detection (and subsequent correction) of loss of positive definiteness of the approximate Hessian matrices (due to rounding errors in finite precision arithmetic) with little extra computational effort. As the theory of Cholesky factorisations is well established (see for example [1, 8, 13]) it is not discussed further here.

This paper examines the performance of a selection of BFGS implementations on a suite of ill-conditioned test problems across a range of dimensions and line search criteria as the precision of second-order information varies from 16 to two digits. The results presented in this paper support those in [9], specifically that there is no numerical evidence to support the claim that a Cholesky factor implementation of the BFGS formula offers any improvement in performance, as is popularly believed, over more straightforward implementations when second-order information is available to full precision. Furthermore, this paper extends these results to show that a factorisation strategy has clear advantages when second-order information is only available to limited precision. However a Cholesky factorisation is not necessarily the best one to use.

2. BFGS formula

The BFGS update formula can be written as

$$B_{k+1} = \left[B + \frac{yy^\mathsf{T}}{s^\mathsf{T} y} - \frac{Bss^\mathsf{T} B}{s^\mathsf{T} B s} \right]_k \tag{2.1}$$

where $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. If the inverse of B_k is denoted by H_k then application of the Sherman-Morrison-Woodbury formula gives

$$H_{k+1} = \left[H + \left(1 + \frac{y^T H y}{s^T y} \right) \frac{ss^T}{s^T y} - \left(\frac{sy^T H + H y s^T}{s^T y} \right) \right]_k. \qquad (2.2)$$

Equation (2.2) allows the direct calculation of the search direction without the need to solve the system of equations (1.1). The implementations discussed in this paper fall into three categories:

- Updates of the approximate Hessian matrices using (2.1).
- Updates of the inverses of the approximate Hessian matrices using (2.2).
- Factorisations: either Cholesky factorisations of the approximate Hessian matrices, or conjugate factorisations of their inverses.

2.1. Conjugate factorisation   The method of conjugate factorisation used in this paper is based on [4], however the idea is not new (see for example [5, 11, 12]). A brief description follows, but see [4] for more details.

The BFGS update formula (2.2) can be written in product form [1] as

$$H_{k+1} = \left[(I - p q^T) H (I - p q^T)^T \right]_k,$$

where

$$q_k = \left[\frac{y}{p^T y} \pm \frac{g}{\sqrt{-p^T g p^T y / \alpha}} \right]_k.$$

If the inverse Hessian approximation matrices are factored so that $H_k = C_k C_k^T$, then the columns of C_k are B_k-conjugate and the search direction is given by $p_k = -C_k d_k$ where elements of $d_k = C_k^T g_k$ are the directional derivatives of f at x_k in the directions of the columns of C_k. The updated conjugate factors can be written as

$$C_{k+1} = \left[C - \frac{p z^T}{p^T y} \mp \frac{p d^T}{\sqrt{-p^T g p^T y / \alpha}} \right]_k \qquad (2.3)$$

where $z_k = C_k^T y_k$ is the difference between the directional derivatives at x_{k+1} and x_k. Then $d_{k+1} = C_{k+1}^T g_{k+1}$ can be written as

$$d_{k+1} = \bar{d}_k - \frac{p_k^T g_{k+1} z_k}{p_k^T y_k} \mp \frac{p_k^T g_{k+1} d_k}{\sqrt{-p_k^T g_k p_k^T y_k / \alpha_k}}, \qquad (2.4)$$

where $\bar{d}_k = C_k^T g_{k+1}$. Equations (2.3) and (2.4) can be written in terms of the new variables d and z so that

$$C_{k+1} = \left[C + \frac{p z^T}{d^T z} \mp \frac{p d^T}{\sqrt{-d^T d d^T z / \alpha}} \right]_k$$

and

$$d_{k+1} = \left[\bar{d} - \frac{d^\mathsf{T} \bar{d} z}{d^\mathsf{T} z} \pm \frac{d^\mathsf{T} \bar{d} d}{\sqrt{-d^\mathsf{T} d d^\mathsf{T} z / \alpha}} \right]_k .$$

There are two obvious implementations, one for each of the $+/-$ signs in (2.3). After limited numerical trials (see [3]) both implementations were found to perform very similarly. The implementation presented in this paper uses the $+$ sign from (2.3).

2.2. Implementations   There are many ways to implement the BFGS formulae presented in (2.1) and (2.2). The four implementations considered in this paper were selected as a representative sample from the 12 BFGS implementations considered in [3]. As all the numerical results were produced using MATLAB, MATLAB's built-in functions were used where convenient. Typewriter font is used to emphasize MATLAB code. The initial Hessian approximation (or its inverse) was set to the identity matrix for each of the implementations.

Bupdate   Uses (2.1) to update the sequence of B_k matrices. The search direction is calculated by using the MATLAB matrix inverse function via the equation $p_k = -\text{inv}(B_k) * g_k$. Note that direct inversion of the B_k matrices is not recommended in practice due to computational expense and inferior numerical stability. It is used here to provide a guideline for the worst performance that would be expected from this type of implementation. However limited numerical trials showed that it performed almost identically to more preferred implementations, using Gaussian elimination, for example.

Hupdate   Uses (2.2) to update the sequence of H_k matrices. The search direction is calculated directly via $p_k = -H_k * g_k$.

Cholesky   Uses a sequence of Cholesky factors L_k which are updated (rather than recomputed from scratch) at each iteration. The particular implementation presented here uses MATLAB's Cholesky factor update command cholupdate. The search direction is calculated with forward and back substitution via $p_k = -L_k^\mathsf{T} \backslash (L_k \backslash g_k)$.

Conjugate   Conjugate factorisation of the inverse approximate Hessian matrices using the plus sign from (2.3). The search direction can then be obtained by direct calculation.

The implementation Bupdate requires $O(n^3)$ operations at each iteration to update the second-order information and compute the new search direction whereas the remaining implementations require only $O(n^2)$ operations. Additionally, the (modified) Cholesky factorisation implementation allows the easy detection of loss of positive definiteness of the approximate Hessian matrices. The other implementations do not have this feature. However with a conjugate factorisation it is extremely unlikely that

the inverse approximate Hessian matrices will lose positive definiteness. The worst that can happen is that they may become positive semi-definite. In fact Powell makes the comment in [12] that:

> We even find that, if we let Z [the conjugate factorisation matrix] be singular initially, then in practice the rounding errors of a sequence of updating calculations remove the singularity very successfully.

Thus if positive definiteness of the inverse approximate Hessian matrices is lost then it is extremely likely it will be restored at the next iteration—or the other way around—it is extremely unlikely that loss of positive definiteness will be maintained for any length of time if conjugate factors are used. Even the unlikely loss of positive definiteness can be detected in a computationally efficient way by using *triangular* conjugate factors. Such factors could be generated and updated at each iteration using a QR-factorisation for example.

3. Numerical results

Each of the four BFGS implementations described above were tested with two different line searches on the suite of 25 test functions listed in Tables 1 and 2 as the precision of the approximate Hessian information varied from 16 to two digits. The varying levels of precision were achieved by truncating the elements of the approximate Hessian matrices (possibly in factored form, or their inverses) to the desired level. For example, the elements of the matrix X are truncated to n digits with $\text{trunc}(X) = 10^{-d} \lfloor 10^d X \rfloor$ where $d = n - \lceil \log_{10}(\max(|X|)) \rceil$.

Each of the higher dimensional tests listed in Table 2 was carried out in 8, 12, 20, 40 and 60 dimensions. The column labelled *Cond* in Table 1 represents the condition number ($C = \|B\|_2 \cdot \|B^{-1}\|_2$) of the Hessian matrices at the solution. Since the condition number is the ratio of the largest singular value to the smallest, all of the Powell singular functions have infinite condition number. Increasing dimension does not alter the condition number of the repeated Rosenbrock functions, and only slightly increases that of the extended Rosenbrock function which has condition number 3.6×10^3 for the 60-d case. The condition numbers of the Hilbert quadratics on the other hand are known to increase dramatically with increasing dimension. The 8-d Hilbert quadratic has condition number 1.5×10^{10} which increases to 1.7×10^{16} in 12 dimensions. More details on the test functions can be found in [9, 10].

A two-sided Wolfe line search was used so that at each iteration α_k was chosen so that $x_{k+1} = x_k + \alpha_k p_k$ satisfies $f_{k+1} \leqslant f_k + \rho \alpha_k p_k^T g_k$ and $|p_k^T g_{k+1}| \leqslant \sigma |p_k^T g_k|$ where the sufficient descent parameter $\rho = 10^{-4}$ and the gradient parameter σ was set to 10^{-3} and 0.9 for what are referred to in the remainder of this paper as *strict* and

TABLE 1. Low dimension test functions.

Function	Dim.	Initial point	Cond.
Rosenbrock	2	$(-1.2, 1)$	2.5×10^3
Powell badly scaled	2	$(0, 1)$	2.1×10^{15}
Repeated Rosenbrock	4	$(-1.2, 1, -1.2, 1)$	2.5×10^3
Extended Rosenbrock	4	$(-1.2, 1, -1.2, 1)$	3.2×10^3
Powell singular	4	$(3, -1, 0, 1)$	∞

TABLE 2. Test functions for 8, 12, 20, 40 and 60 dimensions.

Function	Initial point
Repeated Rosenbrock	$(-1.2, 1, -1.2, 1, \ldots)$
Extended Rosenbrock	$(-1.2, 1, -1.2, 1, \ldots)$
Powell singular	$(3, -1, 0, 1, \ldots)$
Hilbert quadratic	$(0, 0, 0, 0, \ldots)$

standard line searches. The Wolfe line search was implemented using an iterative safeguarded parabolic interpolation scheme.

For each test problem the number of function evaluations, final function value and execution time (in seconds) was recorded. The overall performance of each implementation was determined using the following ranking system. First the implementations were sorted by the number of test functions that were successfully solved. A test problem was deemed to have been successfully solved if the termination criterion $\|\nabla f(x)\| \leqslant 10^{-6}$ was met. If necessary the algorithms were then subsorted by the mean number of function evaluations. Any ties were subsorted by the mean accuracy of the approximations to the minimum function values. The accuracy was measured using $\log_{10}(f - f^*)$ where f^* represents the minimum of the function and f is the final function value. Note that $f^* = 0$ for each of the test problems in Tables 1 and 2. As algorithm execution time depends on the computing environment as well as the implementation, the mean execution times presented here are indicative only, and are not used in the ranking scheme. Only data for the problems that were solved successfully were used in the sorting process.

Practical BFGS implementations may safeguard the positivity condition $s_k^\mathsf{T} y_k > 0$ and only update the approximate Hessian information if $s_k^\mathsf{T} y_k > \epsilon$ for some (generally small) $\epsilon > 0$. However, as it is the "raw" performance of each implementation that is being investigated, the algorithms were terminated whenever they ran into difficulty rather than applying some sort of safeguarding or corrective procedure.

The implementations were deemed unsuccessful and thus terminated if more than 10^5 function evaluations were required, a descent direction was not found, a step of zero length was calculated, the function values became unbounded (as a result of

TABLE 3. Number and type of failures.

Method	Linesearch (strict)	(std)	∞ (strict)	(std)	No descent (strict)	(std)
Hupdate	105	108	1	–	–	–
Bupdate	69	71	20	15	–	–
Cholesky	–	–	49	52	–	–
Conjugate	29	40	14	2	–	2

TABLE 4. Strict line search and 16 digit second-order precision.

Rank	Method	Succ	Fcnt	Accy	Time
1	Hupdate	25	310.4	−13.8	2.6
2	Cholesky	25	321.4	−13.8	2.8
3	Conjugate	25	322.8	−14.1	2.7
4	Bupdate	25	342.4	−14.0	3.1

TABLE 5. Standard line search and 16 digit second-order precision.

Rank	Method	Succ	Fcnt	Accy	Time
1	Cholesky	25	154.9	−12.8	1.6
2	Bupdate	25	156.4	−13.1	1.5
3	Conjugate	25	157.3	−13.1	1.4
4	Hupdate	25	157.9	−13.0	1.3

division by zero due to rounding errors in finite precision arithmetic), a factorisation failed (where appropriate), or the line search failed. The line search failed if the global limit of 10^5 function evaluations was reached, more than 10^3 parabolic interpolation iterations were required, or a zero step was calculated. The number of times each type of failure occurred as the precision of the second-order information varied from 16 to two digits for each of the implementations is presented in Table 3. The columns labelled *Linesearch*, ∞ and *No descent* represent the number of failures that occurred as a result of failure of the line search, division by zero and failure to determine a descent direction. No other types of failure occurred. Furthermore, all of the failures in the line search were caused by the calculation of a zero-step.

All of the implementations presented in this paper were run in a MATLAB R12.1 environment on a Sun-Fire-880 multi-user machine with four 750MHz processors and 8GB of RAM running Solaris 8.

In each of the following result tables the columns labelled *Succ*, *Fcnt*, *Accy* and *Time* represent the number of successfully solved test problems, the mean number of function evaluations, the mean accuracy of the solutions and the mean execution time

TABLE 6. Strict line search and 16–2 digit second-order precision.

Rank	Method	Succ	Fcnt	Accy	Time
1	Conjugate	332	323.1	−13.9	2.4
2	Cholesky	326	361.2	−13.8	2.8
3	Bupdate	286	336.6	−13.7	2.8
4	Hupdate	269	313.0	−13.8	2.6

TABLE 7. Standard line search and 16–2 digit second-order precision.

Rank	Method	Succ	Fcnt	Accy	Time
1	Conjugate	331	159.0	−13.1	1.6
2	Cholesky	323	171.4	−13.0	1.9
3	Bupdate	289	152.1	−12.7	1.7
4	Hupdate	267	150.6	−12.9	1.5

in seconds.

As can be seen from the results presented in Tables 4–7, when successful, all implementations produced similarly accurate approximations to the solutions of the test problems. Furthermore, although the strict line search implementations were slightly more robust, they required nearly double the number of function evaluations as the standard line search implementations. This is a major reason for the popularity of lower accuracy line searches.

3.1. Full precision second-order information The performance of each implementation with full precision (16 digits) second-order information for the strict and standard line searches is presented in Tables 4 and 5.

3.2. Limited precision second-order information The performance of the BFGS implementations as the precision of the second-order information varied from 16 to two digits with the strict and standard line searches is discussed in the following sections. The results are presented in Tables 6 and 7.

Strict line search The number of successfully solved test problems ranged from 332 for Conjugate down to 269 for Hupdate. The mean number of function evaluations ranged from 313.0 for Hupdate through to 361.2 for Cholesky. Overall the mean number of function evaluations was 337 ± 25 and the mean execution times ranged from 2.4 to 2.8 seconds per test problem.

Standard line search The number of successfully solved test problems ranged from 331 for Conjugate down to 267 for Hupdate. The mean number of function evaluations ranged from 150.6 for Hupdate through to 171.4 for Cholesky. Overall
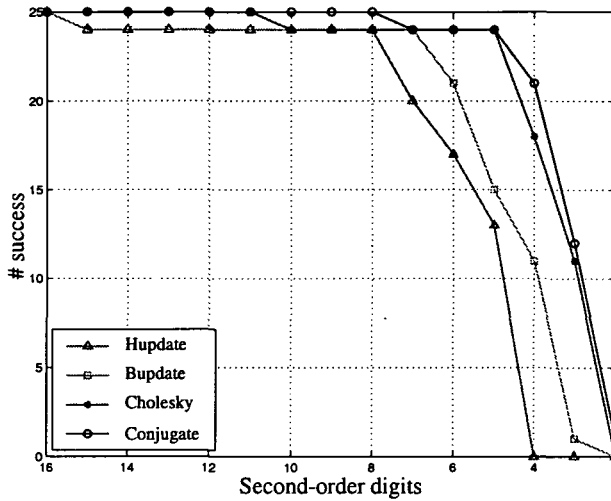
FIGURE 1. BFGS implementations with the standard line search and varying second-order precision.

the mean number of function evaluations was 161 ± 11 and the mean execution times ranged from 1.5 to 1.9 seconds per test problem.

The results in Table 7 are presented graphically in Figure 1. The plot for the strict line search is not presented as it looks very similar. Differences in the BFGS implementations are not noticeable until the precision of the second-order information falls below eight digits. Although not presented here, the equivalent DFP implementations also produced similar results with the strict line search. However, as shown in Figure 2, the equivalent DFP implementations produced noticeably different results with the standard line search. In this case the factorisation implementations produced noticeably better results at single precision (8 digits) than the non-factored implementations [3].

3.3. Quadratic termination For any member of the Broyden family of quasi-Newton methods $B_{n+1} = G$ for any n-dimensional quadratic function with (constant) Hessian matrix G when exact line searches are used [6, pp. 64–65]. Although it is not possible to carry out exact line searches in practice, this result can be used to see how closely each of the above implementations get to the actual Hessian matrix after $n + 1$ iterations. Figure 3 shows $\log_{10} \| H_{n+1} - G^{-1} \|_F$, where $\| \cdot \|_F$ represents the Frobenius norm, for the 4-d Hilbert quadratic and an accurate line search ($\sigma = 10^{-10}$) using the BFGS implementations Bupdate, Cholesky and Conjugate. Note that since the line searches use parabolic interpolation they are exact, except for the errors due to finite precision arithmetic. The difference in norm of the inverse Hessian rather than the Hessian has been used as the inverse Hessian allows the direct calculation
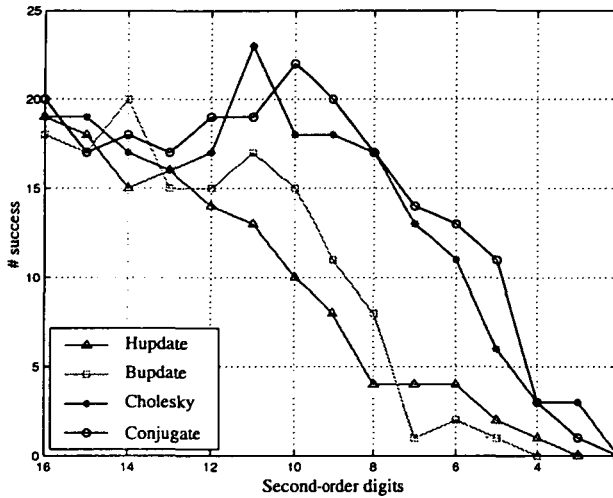
FIGURE 2. DFP implementations with the standard line search and varying second-order precision.

of the search direction, whereas a system of equations must be solved if the Hessian is used. The inverse Hessian is exact but the approximate inverse Hessian matrices H_k are truncated depending on the level of second-order precision. The results for Hupdate clutter the figure somewhat and have been omitted. However if included, the plot for Hupdate would oscillate between the lines for Cholesky and Conjugate.

Note that as the precision of the second-order information falls below about five digits there is a plateau in Figure 3 with a height of about four. The height of this plateau coincides with the norm of the inverse Hessian of the 4-d Hilbert quadratic ($\log_{10} \| G^{-1} \|_F \approx 4.0146$). Presumably once the precision of the second-order information falls below a certain level there is insufficient information to approximate the inverse Hessian to any significant level. Similar results are produced with Hilbert quadratics of different dimensions. In higher dimensions the height of the plateau matches the norm of the inverse Hessian but the plateau starts at higher levels of second-order precision. In lower dimensions the plateau effect is lost and the differences in the performances of the implementations are reduced.

4. Discussion and summary

The performance of four BFGS quasi-Newton implementations on a suite of 25 test functions with two line searches (strict and standard) as the precision of second-order information varied from 16 to two digits have been presented. Although the BFGS implementations with the strict line search were slightly more robust than the BFGS
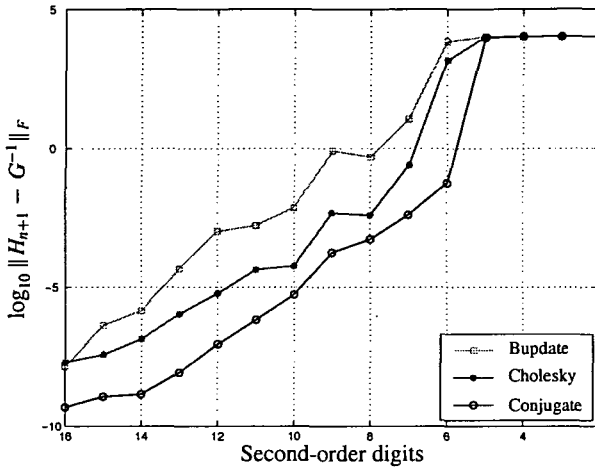
FIGURE 3. Difference in norm after $n + 1$ iterations with varying second-order precision for the 4-d Hilbert quadratic.

implementations with the standard line search they required nearly double the number of function evaluations. When second-order information is available to at least single precision (8 digits) there is no real advantage in any particular implementation.

Cholesky factorisation and triangular conjugate factorisation implementations enable second-order information to be updated and a new search direction computed in $O(n^2)$ operations per iteration as well as allowing the easy detection of loss of positive definiteness of the second-order matrices. However the use of conjugate factorisations eliminates the possibility of negative definiteness or indefiniteness of the inverse approximate Hessian matrices whilst maintaining $O(n^2)$ operations efficiency at each iteration.

The conjugate factorisation implementation produced better approximations to the inverse Hessian matrices of n-dimensional Hilbert quadratics when terminated after $n + 1$ iterations than the other methods. Furthermore as the precision of the second-order information was reduced the conjugate factorisation implementation was able to maintain accurate approximations to the inverse Hessian longer than the other methods.

Figures 1–3 and Tables 6–7 clearly show the importance of a factorisation strategy as the precision of second-order information is reduced. The conjugate factorisation implementation successfully solved more test problems in significantly fewer function evaluations than any of the other implementations, including Cholesky factorisation. It is shown in [2] that grids based on conjugate directions have useful practical and theoretical properties, as such conjugate factorisations should also be of practical importance in a wider optimisation context.

Acknowledgement

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. Math. Appl.* **11** (1973) 73–82.

[2] D. Byatt, I. D. Coope and C. J. Price, "Conjugate grids for unconstrained optimisation", *Comput. Optim. Appl.* **29** (2004), to appear.

[3] D. Byatt, I. D. Coope and C. J. Price, "Performance of various BFGS and DFP implementations with limited precision second order information", Research Report UCDMS2003/1, Univ. of Canterbury, Christchurch, New Zealand, Jan 2003.

[4] I. D. Coope, "A conjugate direction implementation of the BFGS algorithm with automatic scaling", *J. Austral. Math. Soc. Ser. B* **31** (1989) 122–134.

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

[6] R. Fletcher, *Practical methods of optimization*, 2nd ed. (John Wiley & Sons, New York, 1987).

[7] P. E. Gill and W. Murray, "Quasi-Newton methods for unconstrained optimization", *J. Inst. Math. Appl.* **9** (1972) 91–108.

[8] P. E. Gill and W. Murray, "Modification of matrix factorizations after a rank one update", in *The state of the art in numerical analysis* (ed. D. Jacobs), Institute of Mathematics and its Applications, (Academic Press, London, 1976) 55–83.

[9] L. Grandinetti, "Factorization versus non-factorization in quasi-Newtonian algorithms for differentiable optimization", in *Third Symposium on Operations Research (University of Mannheim, Mannheim, 1978)*, Section I, (Hain, Konigstein, 1979) 255–274.

[10] J. J. Moré, B. S. Garbow and K. E. Hillstrom, "Testing unconstrained optimization software", *ACM Trans. Math. Software* **7** (1981) 17–41.

[11] M. R. Osborne and M. A. Saunders, "Descent methods for minimization", in *Optimization* (eds. R. S. Anderssen, L. D. Jennings and D. M. Ryan), (Univ. of Queensland Press, St. Lucia, 1972) 221–237.

[12] M. J. D. Powell, "Updating conjugate directions by the BFGS formula", *Math. Program.* **38** (1987) 29–46.

[13] R. B. Schnabel and E. Eskow, "A revised modified Cholesky factorization algorithm", *SIAM J. Optim.* **9** (1999) 1135–1148.