JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 25, No. 4, AUGUST 1978

Numerical Comparison of Several Variable-Metric Algorithms[1]

D. F. SHANNO[2] AND K. H. PHUA[3]

Communicated by H. Y. Huang

Abstract. The paper compares the numerical performances of the LDL′ decomposition of the BFGS variable-metric algorithm, the Dennis–Mei dogleg algorithm on the BFGS update, and Davidon's projections with the BFGS update with the straight BFGS update on a number of standard test problems. Numerical results indicate that the standard BFGS algorithm is superior to all of the more complex strategies.

Key Words. Unconstrained optimization, LDL′ decomposition, Davidon projections, dogleg algorithm, BFGS algorithm.

1. Introduction

The BFGS algorithm for unconstrained nonlinear optimization, due to Broyden (Ref. 1), Fletcher (Ref. 2), Goldfarb (Ref. 3), and Shanno (Ref. 4), generates a sequence of points x_k to the minimizer $\hat{x}$ of a function $f(x)$, where x is an n-vector and f a real-valued nonlinear function, by

$$x_{k+1} = x_k - \alpha_k H_k g_k, \tag{1}$$

$$H_{k+1} = H_k + (1 + y'_k H_k y_k / p'_k y_k)(p_k p'_k / p'_k y_k) - (H_k y_k p'_k + p_k y'_k H_k)/p'_k y_k, \tag{2}$$

where

$$g_k = \nabla f(x_k), \qquad y_k = g_{k+1} - g_k, \qquad p_k = x_{k+1} - x_k,$$

and α_k is an appropriately chosen scalar.

[1] This research was supported by the National Science Foundation under Research Grant No. MCS77-07327.

[2] Professor, Department of Management Information Systems, College of Business and Public Administration, University of Arizona, Tucson, Arizona.

[3] Lecturer, Department of Computer Science, Nanyang University, Singapore, Malaysia.

In a recent paper (Ref. 5), the authors examined the initial choice H_0 for an approximation to the inverse Hessian, and concluded that, for large problems, initially H_0 can be set to the identity matrix; but, after x_1 has been chosen, but before H_1 has been computed, H_0 should be scaled by

$$\hat{H}_0 = \gamma H_0, \qquad \gamma = p_0' y_0 / y_0' H_0 y_0, \tag{3}$$

and H_1 computed using (2) with $\hat{H}_0$ substituted for H_0.

In that paper, it was shown that, while this scaling was extremely important to increased numerical efficiency on large problems, on small problems (2–4 variables) it sometimes improved, sometimes hurt computational efficiency. A possible explanation for this is that, if γ is small, premultiplying H_0 by γ has the effect of rotating the subsequent search vector p_1 away from the negative gradient g_1. As γ was typically small on those problems where performance deteriorated under scaling, it would appear that a dogleg strategy, which rotates the search vector back toward the gradient, might well improve numerical performance.

In an earlier paper, Shanno, Berg, and Cheston (Ref. 6) showed that, if fairly accurate linear searches were used, the dogleg strategy never improved the performance of the BFGS algorithm. However, using the very inexact searches of the MINIO2 algorithm [Shanno and Phua, Ref. 7], it appeared possible that a dogleg might prove useful.

To this end, we coded the dogleg introduced by Powell (Ref. 8) in the form documented by Dennis and Mei (Ref. 9), but using the BFGS update, rather than the PSB update of Powell (Ref. 8) or the update of Dennis and Mei which incorporates Davidon (Ref. 10) projections. Unfortunately, performance of the algorithm with the scaled BFGS update was uniformly worse than the straightforward scaled BFGS, as the numerical results in Section 3 demonstrate. Thus, we are forced to conclude that the variance in efficiency of the scaled and unscaled BFGS on small problems is indeed random, or at least beyond our power to explain.

Another current hypothesis is that updating an approximation J_k to the Hessian, rather than the approximation H_k to the inverse Hessian, could improve numerical stability and thus numerical efficiency if J_k were updated in the factored LDL′ form where L is lower triangular and D is diagonal. To test this, we implemented the LDL′ update algorithm of Fletcher and Powell (Ref. 11), and tested it with the complementary BFGS update defined by

$$J_{k+1} = J_k + y_k y_k' / p_k' y_k - J_k p_k p_k' J_k / p_k' J_k p_k, \tag{4}$$

where again the algorithm was tested both unscaled and scaled, where the initial scaling was

$$\hat{J}_0 = (1/\gamma) J_0. \tag{5}$$

Here, the scaled and unscaled LDL′ algorithms were so similar to the scaled and unscaled straightforward BFGS in performance that the extra overhead required for the LDL′ algorithm appears not to be justified.

As a further note on this, both the Powell algorithm and the Dennis and Mei algorithm require both H_k and J_k for their dogleg strategies. Our implementation of the Dennis and Mei algorithm used the LDL′ factorization for J_k, and hence required only one matrix to be stored. In view of the discouraging computer results we obtained, which are documented in Section 3, it is doubtful whether a dogleg strategy should ever be employed. If, however, one wishes to use this strategy, this appears to be the most efficient means of implementing it.

Finally, we addressed the question of the numerical efficiency of the update using projections introduced by Davidon (Ref. 10). As this algorithm is fairly complex, and numerically very sensitive to the choice of certain parameters, it is worthwhile to examine in more detail both our implementation and computational experience. This we do in detail in Section 2. We conclude this section by noting simply that, numerically, the Davidon algorithm suffers in comparison to the BFGS algorithm, leading to the conclusion that a straightforward BFGS implementation is both more efficient and more robust than any other known variable-metric algorithm.

2. On the Implementation of Davidon's Projections

In Ref. 10, Davidon introduced three new ideas for variable-metric algorithms, the factored form of the update, the optimally conditioned update, and the use of projections to update in a way that produces finite termination on quadratic object functions. In Ref. 5 it is shown that, in general, the BFGS update is superior to the optimally conditioned update without projections. Turner (Ref. 12) has shown identical performance of the factored and unfactored updates, at least on small problems. It appears likely that this will remain true for large problems. Thus, unless algorithm overhead for one is shown to be superior to the other, the choice of factored versus unfactored for nonprojected algorithms seems open, although preliminary investigation shows that the unfactored algorithm can be implemented in about $3n^2$ multiplications per step, and the factored in $4n^2$.

Thus, the computational question of interest is the use of projections. We address the problem in detail here.

Basically, the Davidon algorithm computes at each iteration the projection operator P defined by

$$P = Y(Y'H^{-1}Y)^{-1}Y'H^{-1}, \tag{6}$$

where Y is the $n \times 2$ matrix defined by

$$Y = [p - Hy, u], \tag{7}$$

and

$$u^* = v'yu - u'yv, \qquad u_0 = H_0 g_0, \qquad v = p - Hy,$$

where we have dropped the subscript k and substituted the superscript $*$ for the subscript $k+1$. Here, H^{-1} is the inverse of any positive-definite update in the Broyden parametric family of updates, where H is the approximation used for the inverse Hessian. The vectors z and γ are then computed by

$$z = Pp, \qquad \gamma = P'y, \tag{8}$$

and H is updated using z and γ in place of p and y. In view of the superiority of the BFGS update, we elected to use this in place of Davidon's optimally conditioned update. Thus, the update that we desire is defined by

$$H^* = H + (1 + \gamma'H\gamma/z'\gamma)(zz'/z'\gamma) - (H\gamma z' + z\gamma'H)/z'\gamma. \tag{9}$$

It remains to demonstrate how to compute z and γ in an efficient manner.

By introducing the factored form for H defined by

$$H = JJ', \tag{10}$$

with update formula

$$J^* = [I + \xi\eta']J \tag{11}$$

for appropriately chosen vectors ξ, η, Davidon was able to define vectors

$$\hat{v} = J^{-1}v, \qquad \hat{u} = J^{-1}u; \tag{12}$$

and, by setting

$$\hat{Y} = [\hat{v}, \hat{u}],$$

trivial computation yields

$$\hat{Y}'\hat{Y} = Y'H^{-1}Y. \tag{13}$$

While use of this factorable form appears to be the most efficient means of implementing this algorithm, a straightforward means of using the update (9) can also be devised at the cost of carrying another vector. To see this, note that, as

$$p = -\alpha H g,$$

we have

$$H^{-1}v = H^{-1}(p - Hy) = -\alpha g - y, \tag{14}$$

and that

$$H_0^{-1}u_0 = H_0^{-1}H_0 g_0 = g_0.$$

We now define

$$s = H^{-1}u, \tag{15}$$

and consider the problem of calculating s^*. As

$$J^* = H^{*-1},$$

where J^* is defined by (4), we have

$$s^* = J^*u^* = J^*(v'yu - u'yv) = v'yJ^*u - u'yJ^*v. \tag{16}$$

Now, applying (4), we get

$$\begin{aligned} J^*u &= Ju + (y'u/y'p)y - Jpp'Ju/p'Jp \\ &= s + (y'u/y'p)y - (p's/p'g)g, \end{aligned} \tag{17}$$

and

$$\begin{aligned} J^*v &= Jv + (y'v/y'p)y - Jpp'Jv/p'Jp \\ &= -\alpha g - y + (y'v/y'p)y - [p'(-\alpha g - y)/p'g]g \\ &= (y'v/y'p - 1)y - [\alpha + p'(-\alpha g - y)/p'g]g. \end{aligned} \tag{18}$$

Thus, we have, from (16)–(18),

$$\begin{aligned} s^* &= v'ys + [v'yy'u/y'p - y'u(y'v/y'p - 1)]y \\ &\quad + \{-u'y[\alpha + p'(-\alpha g - y)/p'g] - v'y(p's/p'g)\}g \\ &= v'ys + u'yy + [(p'yu'y - v'yp's)/p'g]g. \end{aligned} \tag{19}$$

Then, if we define

$$t = H^{-1}v,$$

we have

$$(Y'H^{-1}Y) = \begin{bmatrix} v't & v's \\ u't & u's \end{bmatrix}, \tag{20}$$

and

$$(Y'H^{-1}Y)^{-1}=(1/d)\begin{bmatrix} u's & -v's \\ -u't & v't \end{bmatrix}, \tag{21}$$

where

$$d=v'tu's-v'su't. \tag{22}$$

Now, we have

$$Pp=Y(Y'H^{-1}Y)^{-1}Y'H^{-1}p=-\alpha Y(Y'H^{-1}Y)^{-1}Y'g, \tag{23}$$

and

$$P'y=H^{-1}Y(Y'H^{-1}Y)^{-1}Y'y. \tag{24}$$

Then, since

$$Y'y=\begin{pmatrix} v'y \\ u'y \end{pmatrix}, \qquad H^{-1}Y=(t,s),$$

(24) yields

$$\begin{aligned}\gamma=P'y&=(1/d)(t,s)\begin{bmatrix} u's & -v's \\ -u't & v't \end{bmatrix}\begin{pmatrix} v'y \\ u'y \end{pmatrix}\\ &=(1/d)[(u'sv'y-v'su'y)t+(v'tu'y-u'tv'y)s]. \end{aligned} \tag{25}$$

Similarly,

$$\begin{aligned}z=Pp&=-(\alpha/d)(v,u)\begin{bmatrix} u's & -v's \\ -u't & v't \end{bmatrix}\begin{pmatrix} v'g \\ u'g \end{pmatrix}\\ &=-(\alpha/d)[(u'sv'g-v'su'g)v+(v'tu'g-u'tv'g)u]. \end{aligned} \tag{26}$$

Also, in the event that u and v are colinear (which always happens on the last step for quadratic functions), P is simply defined by

$$P=v(v'H^{-1}v)^{-1}v'H^{-1}, \tag{27}$$

so we have

$$z=Pp=-(\alpha v'g/v't)v \tag{28}$$

and

$$\gamma=P'y=(v'y/v't)t. \tag{29}$$

It is this algorithm which we implemented and tested. It should be noted that a similar implementation can be derived for any member of the Broyden family by using the proper update formula for J^*.

Several further points on this algorithm are worthy of discussion. As mentioned previously, we have found the algorithm highly sensitive to numerical problems caused by roundoff. Specifically, if $p'y$ is small, projecting p and y leads to serious stability problems. This was also noted by Turner. Thus, we include a parameter ϵ in our algorithm, and test if

$$p'y < \epsilon.$$

If so, we update without projecting and reset

$$u = Hg.$$

Also, Davidon noted that, if $z'\gamma$ is small, it is safer not to use z and γ, but rather to use p and y. Thus, if

$$z'\gamma < \epsilon,$$

we update using p and y and reset

$$u = Hg.$$

Again, u can become very small at a point well away from the minimizer $\hat{x}$; so, if

$$u'u < \epsilon,$$

we reset u to Hg before projecting. This also was suggested by Davidon.

He also suggested skipping updating altogether if

$$v^1 H^{-1} v < \epsilon,$$

and we include this in our algorithm.

Finally, his test on whether to do a rank-one or rank-two projection depends on how close d is to zero. He suggests using a rank-two update unless

$$d \leq 10^6 |u't - v's|. \tag{30}$$

Because

$$u't = u's$$

with exact arithmetic, this requires d to be very small.

Davidon tested only problems of dimension four or less in his paper; and, for these problems, this test is satisfactory. However, for larger problems, this rapidly becomes very unsatisfactory, leading to infinitesimally small search directions well away from the minimum. Consequently, we have incorporated the rule that a rank-two projection is done unless

$$d \leq \epsilon^2, \tag{31}$$

at which point a rank one is done.

We have experimented with various values of ϵ; and, using double-precision arithmetic on the DEC 10 computer, have found $\epsilon = 10^{-10}$ to be most satisfactory, although no ϵ-value proved ideal. This will be discussed in more detail in the next section.

3. Numerical Results

Seven algorithms were coded and tested. Six used the MINIO2 algorithm for steplength determination [see Shanno and Phua, Ref. 7]. These were the BFGS, LDL′, and projected BFGS (PBFGS) update algorithms, each done with and without initial scaling of the matrix H_0. The dogleg algorithm was implemented exactly as described by Dennis and Mei (Ref. 9), with the aforementioned differences that the BFGS update in LDL′ form was used.

The six test functions used are documented in Ref. 5. Briefly, they are the extended Rosenbrock function for $n = 2$ and $n = 20$, Wood's function, the Weibull function, Oren's power function with $n = 20$, the extended Powell function with $n = 4$ and $n = 36$, and the Mancino function with $n = 10$, 20, and 30.

In all cases, convergence was determined when each element of the gradient vector was less than 10^{-5}. In the tables, ITER is the number of matrix updates performed, while IFUN is the number of function and gradient calls. One reason for the larger number of function calls per iteration on average for PBFGS is that, whenever an update is skipped, we do not count this as a major iteration. Whether this is the best accounting possible is debatable; but, if the update overhead is considered significant, which it is on all of these functions except the Weibull and Mancino functions, it appears to be the fairest measure of performances.

Also, in all cases, the initial estimates to x_0 are included in the tables, with the exception of the Mancino function. Those initial estimates are documented in Ref. 5.

In Tables 1–2, the asterisk indicates that the method had failed to converge in the indicated number of iterations. The (1) for the dogleg strategy on the Weibull function results from the algorithm determining that each initial estimate was a minimum, where indeed one is a maximum and others have bad numerical gradient problems. This can be overcome by forcing the method to search the initial vector more carefully. However, comparison of the results with the pure scaled BFGS make it dubious that the strategy should be pursued further.

To indicate the ϵ-sensitivity of PBFGS, as previously indicated, all of these results use

$$\epsilon = 10^{-10}.$$

Table 1. Numerical results.

Function	LDL′ unscaled		LDL′ scaled		Dogleg scaled		PBFGS unscaled		PBFGS scaled		BFGS unscaled		BFGS scaled	
Initial point	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN
Rosenbrock 2														
(−1.2, 1)	35	42	35	42	53	69	29	36	35	42	35	42	35	42
(2, −2)	12	14	38	47	31	44	17	19	39	49	16	18	38	47
(−3.635, 5.621)	47	59	49	61	50	61	47	61	47	58	46	55	50	62
(6.39, −0.221)	54	64	29	34	50	59	47	59	24	28	54	64	29	34
(1.489, −2.547)	22	25	29	35	36	49	21	25	30	39	21	24	29	35
Rosenbrock 20														
(−1.2, 1)	146	191	35	42	57	71	126	301*	37	44	129	173	35	42
(2, −2)	38	46	38	47	31	44	26	35	37	48	34	42	38	47
(−3.635, 5.621)	162	189	49	61	50	61	162	301*	78	301*	155	175	50	62
(6.39, −0.221)	182	202	29	34	50	59	113	301*	23	26	122	137	29	34
(1.489, −2.547)	62	78	29	35	36	49	51	73	28	26	85	100	29	35
Wood														
(−3, −1, −3, −1)	80	94	38	42	46	55	73	98	28	36	79	98	38	42
(−3, 1, −3, 1)	84	103	93	113	41	50	89	111	71	134	79	102	95	112
(−1.2, 1, −1.2, 1)	60	77	82	104	27	200*	68	91	74	114	69	89	85	111
(−1.2, 1, 1.2, 1)	36	45	44	51	51	69	37	47	42	51	36	48	44	51

* Method failed to converge in the indicated number of iterations.

Table 2. Numerical results.

Function / Initial point	LDL′ unscaled		LDL′ scaled		Dogleg scaled		PBFGS unscaled		PBFGS scaled		BFGS unscaled		BFGS scaled	
	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN	ITER	IFUN
Weibull														
(5, 0.15, 2.5)	38	43	49	60	1	(1)	28	40	46	52	37	54	49	57
(250, 0.3, 5)	45	61	59	75	1	(1)	41	55	57	78	39	56	57	77
(100, 3, 12.5)	30	50	45	57	1	(1)	40	56	41	52	42	49	45	57
Oren 20														
(1, 1, . . . , 1)	160	173	193	203	188	200	117	301*	116	301*	96	100	232	234
Powell 4														
(−3, −1, 0, 1)	41	43	59	61	54	63	30	34	33	201*	37	39	59	61
Powell 36														
(−3, −1, 0, 1, . . .)	83	87	59	61	54	63	55	79	39	41	87	89	59	61
Mancino														
Mancino 10														
(. . . . . .)	22	28	8	10	7	19	19	34	4	8	18	28	5	8
Mancino 20														
(. . . . . .)	35	47	8	10	8	18	29	56	7	10	35	47	8	10
Mancino 30														
(. . . . . .)	52	69	8	10	7	16	29	62	8	12	52	69	8	10

* Method failed to converge in the indicated number of iterations.

Table 3

n	ITER
4	4
6	7
8	11
10	13
20	22
40	33

However, when ϵ is changed to 10^{-20}, while most results get worse, convergence occurs on the four-dimensional Powell function in 37 iterations and 39 function and gradient evaluations. Also, finite termination on a quadratic function is also ϵ-sensitive. We tested PBFGS scaled on

$$f(x) = \sum_{i=1}^{n} ix_i^2,$$

with initial estimates

$$x_i = 1,$$

and obtained the number of iterations necessary for convergence with $\epsilon = 10^{-10}$ as shown in Table 3. These results change with ϵ but demonstrate that, even for a very well-conditioned quadratic function, the method is highly ϵ-sensitive.

While the results of these tests make it dubious that the extra overhead of calculating the projections is worthwhile, in view of performance, more testing may show classes of functions where the method is useful. In this event, on all large problems, we strongly recommend initial scaling, as the extended Powell, extended Rosenbrock, and Mancino results clearly demonstrate.

References

1. BROYDEN, C. G., *The Convergence of a Class of Double Rank Minimization Algorithms 2, the New Algorithm*, Journal of the Institute of Mathematics and Applications, Vol. 6, pp. 222–231, 1970.
2. FLETCHER, R., *A New Approach to Variable Metric Algorithms*, Computer Journal, Vol. 13, pp. 317–322, 1970.

3. GOLDFARB, D., *A Family of Variable Metric Algorithms Derived by Variational Means*, Mathematics of Computation, Vol. 24, pp. 23–26, 1970.
4. SHANNO, D. F., *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computation, Vol. 24, pp. 647–656, 1970.
5. SHANNO, D. F., and PHUA, K. H., *Matrix Conditioning and Nonlinear Optimization*, Mathematical Programming, Vol. 14, pp. 149–160, 1978.
6. SHANNO, D. F., BERG, A., and CHESTON, G., *Restarts and Rotations of Quasi-Newton Methods*, Information Processing 74, Edited by J. L. Rosenfeld, North-Holland Publishing Company, Amsterdam, Holland, 1974.
7. SHANNO, D. F., and PHUA, K. H., *Minimization of Unconstrained Multivariate Functions*, ACM Transactions on Mathematical Software, Vol. 2, pp. 87–94, 1976.
8. POWELL, M. J. D., *A New Algorithm for Unconstrained Optimization*, Nonlinear Programming, Edited by J. B. Rosen, O. C. Mangasarian, and K. Ritter, Academic Press, New York, New York, 1970.
9. DENNIS, J. E., and MEI, H. H. W., *An Unconstrained Optimization Algorithm Which Uses Function and Gradient Values*, Cornell University, Computer Science Department, Technical Report No. 75-246, 1975.
10. DAVIDON, W. C., *Optimally Conditioned Optimization Algorithms Without Line Searches*, Mathematical Programming, Vol. 9, pp. 1–30, 1975.
11. FLETCHER, R., and POWELL, M. J. D., *On the Modification of LDL' Factorizations*, Atomic Energy Research Establishment, Harwell, England, Report No. HL73/6036, 1973.
12. TURNER, P. R., *The Use of Projections and Factorization in Optimization Algorithms*, University of Lancaster, Lancaster, England, Department of Mathematics, Working Paper, 1977.