Numerical Algorithms 22 (1999) 99–112

Improved Hessian approximations for the limited memory BFGS method *

Mehiddin Al-Baali

Department of Mathematics and Statistics, Sultan Qaboos University, P.O. Box 36, Al-Khod 123, Muscat, Sultanate of Oman
E-mail: albaali@squ.edu.om

Received 29 May 1998; revised 14 April 1999
Communicated by C. Brezinski

This paper considers simple modifications of the limited memory BFGS (L-BFGS) method for large scale optimization. It describes algorithms in which alternating ways of re-using a given set of stored difference vectors are outlined. The proposed algorithms resemble the L-BFGS method, except that the initial Hessian approximation is defined implicitly like the L-BFGS Hessian in terms of some stored vectors rather than the usual choice of a multiple of the unit matrix. Numerical experiments show that the new algorithms yield desirable improvement over the L-BFGS method.

Keywords: large scale optimization, quasi-Newton methods, BFGS updating formula, limited memory BFGS method

AMS subject classification: 65, 49, 90C30

1. Introduction

This paper considers minimizing the objective function $f(x)$, where $x \in \mathbb{R}^n$. It is assumed that n is large and the gradient vector $g(x) = \Delta f(x)$ is available, but the Hessian matrix and its approximation cannot be computed or stored. A useful method for solving this type of large scale optimization problems is the limited memory BFGS (L-BFGS) method of Nocedal [14], because of its simplicity and low storage requirement (see also Liu and Nocedal [11]). Although a Hessian approximation of the objective function is required at each iteration, L-BFGS does not store it explicitly. Instead, a certain user's number (say, $2m$) of vectors are used to store the information of the Hessian approximation. However, for a sufficiently small value of m, L-BFGS suffers on some (particularly, ill-conditioned) problems (e.g., Byrd et al. [6]). To avoid this difficulty, this paper proposes modifications of the L-BFGS method with the aim of getting improved performance from a given set of the stored vectors.

* An early version of this work was presented at the 16th biennial conference on Numerical Analysis, Dundee, UK, 1995.

The L-BFGS method falls into the class of line search quasi-Newton methods of the form

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}, \tag{1}$$

where $x^{(1)}$ is given, $\alpha^{(k)}$ is a steplength parameter and

$$s^{(k)} = -H^{(k)} g^{(k)} \tag{2}$$

is the search direction. Here $g^{(k)}$ denotes the gradient $g(x^{(k)})$ and $H^{(k)}$ approximates the inverse Hessian matrix at $x^{(k)}$. For a given $H^{(1)}$, $H^{(k)}$ is updated for which the quasi-Newton condition

$$H^{(k+1)} \gamma^{(k)} = \delta^{(k)}, \tag{3}$$

where

$$\delta^{(k)} = x^{(k+1)} - x^{(k)} \quad \text{and} \quad \gamma^{(k)} = g^{(k+1)} - g^{(k)}, \tag{4}$$

is satisfied.

In the standard BFGS method, $H^{(k)}$ is updated to

$$H^{(k+1)} = \text{bfgs}\big(H^{(k)}, \delta^{(k)}, \gamma^{(k)}\big), \tag{5}$$

where the function

$$\text{bfgs}(H, \delta, \gamma) = \left(I - \frac{\delta \gamma^{\mathrm{T}}}{\delta^{\mathrm{T}} \gamma}\right) H \left(I - \frac{\gamma \delta^{\mathrm{T}}}{\delta^{\mathrm{T}} \gamma}\right) + \frac{\delta \delta^{\mathrm{T}}}{\delta^{\mathrm{T}} \gamma} \tag{6}$$

is the BFGS updating formula. The BFGS method is robust and attractive for solving optimization problems (e.g., Fletcher [7]). Because it is assumed that storage for the BFGS Hessian (5) is not available when the number of variables is sufficiently large, limited memory methods have been proposed to define other Hessian approximations (e.g., Liu and Nocedal [11]). In this paper we consider the following type of methods.

The L-BFGS method resembles the BFGS method, except that the Hessian approximation $H^{(k+1)}$ is defined implicitly as the outcome of updating a suitable matrix $D^{(k)}$

$$\widehat{m} = \min(m, k) \tag{7}$$

times in terms of the difference vector pairs

$$\big\{\delta^{(i)}, \gamma^{(i)}\big\}, \quad k - \widehat{m} + 1 \leqslant i \leqslant k, \tag{8}$$

which are stored during the previous $\widehat{m}$ iterations. Note that at each iteration, once $k > m$, the oldest is deleted from this sequence and is replaced by the newest pair. The order of the updates is chosen so that the Hessian approximation satisfies the quasi-Newton condition (3). In particular,

$$H^{(k+1)} = \text{lbfgs}\big(\widehat{m}, D^{(k)}, \delta^{(k)}, \ldots, \delta^{(k-\widehat{m}+1)}, \gamma^{(k)}, \ldots, \gamma^{(k-\widehat{m}+1)}\big), \tag{9}$$

where for any $i \geqslant j$,

$$\begin{aligned}&\text{lbfgs}\big(j, D, \delta^{(i)}, \ldots, \delta^{(i-j+1)}, \gamma^{(i)}, \ldots, \gamma^{(i-j+1)}\big)\\ &\quad = \text{bfgs}\big(\text{lbfgs}\big(j-1, D, \delta^{(i-1)}, \ldots, \delta^{(i-j+1)}, \gamma^{(i-1)}, \ldots, \gamma^{(i-j+1)}\big), \delta^{(i)}, \gamma^{(i)}\big) \quad (10)\end{aligned}$$

is the L-BFGS updating formula which employs j BFGS updates initiated by

$$\text{lbfgs}(1, D, \delta, \gamma) = \text{bfgs}(D, \delta, \gamma). \quad (11)$$

If $D^{(k)}$ (referred to as the *basic matrix*) is chosen diagonal, Nocedal [14] shows that the next search direction (defined by the product in (2) with k replaced by $k+1$) can be computed in terms of the basic matrix and the stored pairs (8) without calculating $H^{(k+1)}$ explicitly. See also, for example, Averick and Moré [5] where the product has been calculated using a pseudocode.

Since the user specifies a small amount of storage to be used by giving a small number m (usually $3 \leqslant m \leqslant 7$) which is significantly smaller than n, the choice of a basic matrix plays an important role in the algorithm performance. For the recommended choice of a multiple of the unit matrix of Liu and Nocedal [11] the L-BFGS method can be very slow on certain type of problems, though it works well on general large scale optimization problems (e.g., [1,5,6,8,10,11,13,20]).

The main aim of this paper is to propose a technique for calculating a basic matrix which has some properties more desirable than that of Liu and Nocedal [11]. Since the stored pairs contain information about the curvature of the function, it seems useful to use them further to calculate not only the L-BFGS Hessian (9) but also a basic matrix. This can be done in various ways.

Indeed Liu and Nocedal [11] define their recommended basic matrix in terms of the most recent pair of the sequence (8). In section 2 we modify this choice so that it still remains in the form of a multiple of the unit matrix, but depends on both the most recent and oldest pairs of the sequence (8).

As the memory is limited and since the basic matrix plays the role of initial Hessian approximation which is not necessarily diagonal, we prefer to define $D^{(k)}$ implicitly in a way similar to the L-BFGS Hessian (9), but by employing a number (say, p) of updates in terms of p difference vectors. In this way the Hessian approximation is maintained in the L-BFGS Hessian form which has the desirable feature that the search direction is still computed using a pseudocode.

This modified technique is similar to that of Byrd et al. [6] who generated p pairs by the inner conjugate gradient iterations of one step of the discrete-truncated Newton method applied at certain iterations. However, for simplicity, this paper considers some p pairs from the sequence (8). We assume that the order of the p updates is made so that the resulting matrix (say, H^+) satisfies the quasi-Newton condition (3) with $H^{(k+1)}$ replaced by H^+. Thus, this matrix (referred to as *basic Hessian*) defines an initial Hessian approximation preferable to a diagonal matrix.

In section 3, we precisely define some basic Hessian H^+. We also consider a similar technique of modification which defines the Hessian approximation by updating

the L-BFGS Hessian (9) in terms of the p pairs (i.e., the m and p updates are reversed in the previous technique so that (9) becomes a basic Hessian). This approach follows from the modified BFGS method of Al-Baali [3], but with the BFGS Hessian replaced by the L-BFGS Hessian. Because the author reported encouraging results, we expect that the new technique would work well in practice.

Indeed the above two techniques improve over the L-BFGS method as shown in section 4, which discusses some numerical results required to solve a set of standard test (including ill-conditioned) problems. Section 4 also shows that further improvement can be obtained when the number of updates exceeds $m + p$ in a way to be defined. It is concluded that several new algorithms are desirable in practice.

Finally, section 5 gives some concluding remarks.

2. Improved basic matrix approximations

Three possible choices of the basic matrix of the simple form $D^{(k)} = \nu I$, where I is the identity matrix and ν is a parameter, are described in this section.

For the choice $\nu = 1$, the L-BFGS formula (9) starts updating $D^{(k)} = I$ in terms of the oldest pair of the sequence of the stored vectors (8). Thus it is possible to scale the identity matrix before updating, using the scaling technique of Shanno and Phua [16], in which the basic matrix becomes

$$D^{(k)} = \nu_0^{(k)} I, \quad \nu_0^{(k)} = \frac{\delta^{(k-\widehat{m}+1)T}\gamma^{(k-\widehat{m}+1)}}{\gamma^{(k-\widehat{m}+1)T}\gamma^{(k-\widehat{m}+1)}}. \tag{12}$$

To incorporate the up-to-date information, Liu and Nocedal [11] replaced this basic matrix by

$$D^{(k)} = \nu_1^{(k)} I, \quad \nu_1^{(k)} = \frac{\delta^{(k)\,\mathrm{T}}\gamma^{(k)}}{\gamma^{(k)\,\mathrm{T}}\gamma^{(k)}}, \tag{13}$$

which depends only on the most recent stored pair. These authors reported that generally the latter choice is preferable to the former.

Since the scalar $\nu_1^{(k)}$ is not related to the information required to perform the first update of the L-BFGS formula, it does not scale the identity matrix in the sense of the self-scaling methods (e.g., Oren and Luenberger [15]). Instead, we may state that the basic matrix (13) provides an initial Hessian approximation preferable to that given in (12). Therefore, we can scale the "good" basic matrix ($\nu_1^{(k)} I$) before updating it. Using, in particular, the modified scaling technique of Al-Baali [4], we obtain the scaling factor

$$\nu^{(k)} = \max\left(\frac{\delta^{(k-\widehat{m}+1)\mathrm{T}}\gamma^{(k-\widehat{m}+1)}}{\gamma^{(k-\widehat{m}+1)\mathrm{T}}(\nu_1^{(k)} I)\gamma^{(k-\widehat{m}+1)}}, 1\right) = \max\left(\frac{\nu_0^{(k)}}{\nu_1^{(k)}}, 1\right). \tag{14}$$

Thus we update $\nu^{(k)}(\nu_1^{(k)}I)$ $(= \nu_2^{(k)}I$, say) rather than $(\nu_1^{(k)}I)$, i.e., we let

$$D^{(k)} = \nu_2^{(k)}I, \quad \nu_2^{(k)} = \max\bigl(\nu_0^{(k)}, \nu_1^{(k)}\bigr). \tag{15}$$

In practice, Al-Baali [1] reported that there is little to choose between both basic matrices (13) and (15). This might be because of his experiment which was based on a small number of test problems. When we applied the L-BFGS method to various test problems, we observed that generally the latter choice is better than the former one (for further details, see section 4).

3. Improved basic Hessian approximations

Other modified techniques define basic matrices by some initial Hessian approximations. For $m \geqslant 2$, we first replace the basic matrix $D^{(k)}$ appearing in (9) by some basic Hessian H^+, i.e., we let the Hessian approximation be defined by

$$H^{(k+1)} = \text{lbfgs}\bigl(\widehat{m}, H^+, \delta^{(k)}, \ldots, \delta^{(k-\widehat{m}+1)}, \gamma^{(k)}, \ldots, \gamma^{(k-\widehat{m}+1)}\bigr). \tag{16}$$

There are various choices for H^+. In particular, we consider the $\hat{p}$ updates

$$H^+ = \text{lbfgs}\bigl(\hat{p}, D^{(k)}, \delta^{(k)}, \ldots, \delta^{(k-\hat{p}+1)}, \gamma^{(k)}, \ldots, \gamma^{(k-\hat{p}+1)}\bigr), \tag{17}$$

which depend on $\{\delta^{(i)}, \gamma^{(i)}\}_{i=k-\hat{p}+1}^{k}$ (a subsequence of (8)) and a basic matrix $D^{(k)}$ defined by some formula in section 2. For a given $p \leqslant m$, we let

$$\hat{p} = \min\bigl(p, k - \widehat{m}\bigr), \tag{18}$$

which follows from (7) with m replaced by $m+p$. Note that when $\hat{p} = 0$, we assume that the L-BFGS formula (17) is not employed (i.e., we let $H^+ = D^{(k)}$).

Equation (18) ensures that formula (17) becomes active once $k > m$ and that the total number of updates $\widehat{m} + \hat{p} \leqslant k$ reaches its maximum $m+p$ whenever $k \geqslant m+p$. Although it is possible to choose $\hat{p} = \widehat{m}$, the choice (18) ensures that the total number of updates remains (as for L-BFGS) smaller than or equal to k and that the first m iterations are identical to those of the L-BFGS method (as well as the BFGS method if the basic matrix is defined by (12)). Note that this modified L-BFGS algorithm was proposed earlier by Al-Baali [1], who applied it with $m \leqslant 5$ and $p = 1$ to a set of a small number of test problems. It is also similar to the method of Byrd et al. [6], except that the $\hat{p}$ pairs are generated by employing one step of a certain method at certain iterations. Thus the storage becomes $2(m + \hat{p})$ vectors rather than $2m$.

We now consider another modification of the L-BFGS Hessian (9). We let the Hessian approximation be defined as above except that $\widehat{m}$ and $\hat{p}$, appearing respectively in (16) and (17), are interchanged. This technique is similar to that of Al-Baali [3], except that the BFGS Hessian is replaced by the L-BFGS Hessian.

Since the above modified algorithms for some values of m and p seem (see section 4) to improve the performance of L-BFGS as p increases, it is worth employing

further updates based on the following observation. The L-BFGS method corrects the basic matrix to a Hessian approximation, while the above modified algorithms correct a basic Hessian approximation to a new Hessian approximation (some formulas are given by Al-Baali [2]). Similarly, we may repeat some updates several times. In section 4, we consider some algorithms which employ $p + 1$ corrections. Although the order of the updates can be defined in various ways, we rely on the recent pairs of the sequence (8) more than the older ones, since usually the former pairs have better information about the curvature of the function. For example, Liu and Nocedal [11] reported that the choice (13) works better than the choice (12), which depend on the most recent and the oldest pairs of the sequence (8), respectively.

Finally, it is worth noting that the R-linear convergence result for the L-BFGS method obtained under mild conditions by Liu and Nocedal [11] is still valid for the modified L-BFGS methods considered here. The reason is that the new updating formulas can be written like the L-BFGS formula, except that the basic matrix is replaced by an L-BFGS Hessian form rather than a diagonal matrix.

4. Numerical experiments

Let D indicate a given basic matrix, $\mathrm{L}_m H$ an application of the L-BFGS formula (9) with $D^{(k)}$ replaced by H, and $R_p = \mathrm{L}_p \mathrm{L}_{p-1} \cdots \mathrm{L}_1 \mathrm{L}_0$, where $\mathrm{L}_0 H = H$. For given D, m and $p \leqslant m$, we can list the methods used in our tests in the following way:

1. $\mathrm{L}_m D$. This is the usual L-BFGS method of Nocedal [14] (as defined by (9)).
2. $\mathrm{L}_m \mathrm{L}_p D$. This is defined by (16) and (17).
3. $\mathrm{L}_p \mathrm{L}_m D$. This is similar to $\mathrm{L}_m \mathrm{L}_p D$, but the L_m and L_p updates are reversed (see section 3 for details).
4. $\mathrm{L}_m R_p D$. This is based on (16) and (17).
5. $R_p \mathrm{L}_m D$. This is like $\mathrm{L}_m R_p D$, but the L_m and R_p updates are reversed.

It is clear that all these limited memory methods always begin by updating a basic matrix D defined by either (13) or (15) (referred to as D_1 and D_2, respectively). These algorithms are identical if $p = 0$. Note that the modes $\mathrm{L}_1 \mathrm{L}_m$ and $R_1 \mathrm{L}_m$ reduce to L_m, while $\mathrm{L}_m \mathrm{L}_1$ and $\mathrm{L}_2 \mathrm{L}_m$ are equivalent to $\mathrm{L}_m R_1$ and $R_2 \mathrm{L}_m$, respectively.

For $m \geqslant 2$ and $p \geqslant 1$, table 1 gives the number of times that the BFGS updates are employed per each iteration sufficiently large. At every iteration, m is replaced by $\widehat{m}$ which is given by (7) and p by $\hat{p}$, so that the total number of updates $\widehat{m} + \hat{p}$ is less than or equal to k. For the modes $\mathrm{L}_m \mathrm{L}_p$ and $\mathrm{L}_p \mathrm{L}_m$, $\hat{p}$ is defined by (18), while for $\mathrm{L}_m R_p$ and $R_p \mathrm{L}_m$ it is given by

$$\hat{p} = \big\{ j\colon\ j = \max\big(\min\big(1, k - \widehat{m}\big), \ldots, \min\big(p, k - \widehat{m}\big)\big),\ j(j+1)/2 \leqslant k - \widehat{m} \big\}. \quad (19)$$

Table 1
Number of updates per each iteration sufficiently large.

L_m	$\mathrm{L}_m\mathrm{L}_p$	$\mathrm{L}_p\mathrm{L}_m$	$\mathrm{L}_m R_p$	$R_p\mathrm{L}_m$
m	$m+p$	$m+p-1$	$m+p(p+1)/2$	$m+p(p+1)/2-1$

It is worth noting that in some experiments we replaced the choice (19) by $\hat{p} = \widehat{m}$. We observed that both choices work similarly. Since the latter choice yields more updates than the former one, we consider (19) only in this section. We also tested other algorithms depending on other orders of updates. But we do not include them here because it is hoped that the above algorithms give a sufficient basis to illustrate the idea of re-using some vector pairs which belong to the sequence (8).

Our experiments were performed in double precision with machine $\varepsilon = 2^{-52}$ ($\approx 2.22\times 10^{-16}$) using a software routine that implements all the algorithms considered here. The routine is a modified version of Fletcher's that implements the standard L-BFGS method of Liu and Nocedal [11] (indicated by $\mathrm{L}_m D_1$). Thus the algorithms used in our implementation differ only in the choices of D, the values of the fixed numbers m and p and another number used to select one of the above methods. The line search subroutine finds a steplength $\alpha^{(k)}$ that satisfies the strong Wolfe conditions

$$f^{(k+1)} \leqslant f^{(k)} + c_1\delta^{(k)T}g^{(k)}, \qquad \left|\delta^{(k)T}g^{(k+1)}\right| \leqslant -c_2\delta^{(k)T}g^{(k)}, \tag{20}$$

where $f^{(k)}$ denotes $f(x^{(k)})$, with the choices $c_1 = 10^{-4}$ and $c_2 = 0.9$. The runs were terminated when both

$$f^{(k)} - f^{(k+1)} \leqslant \varepsilon \max\left(1, \left|f^{(k+1)}\right|\right) \tag{21}$$

and

$$\left\|g^{(k+1)}\right\| \leqslant \tilde{\varepsilon} \max\left(1, \left|f^{(k+1)}\right|\right), \tag{22}$$

where $\|\cdot\|$ denotes the usual Euclidean norm and $\tilde{\varepsilon} = 10\sqrt{\varepsilon}$ ($\approx 1.47\times 10^{-7}$). We use this stopping condition as a fair criterion for comparing different algorithms. All the runs achieved essentially the same accuracy.

For some values of the fixed numbers, we tested the above listed algorithms on two sets of 28 quartic functions and 27 general problems. The results are discussed in the next subsections.

4.1. Applications to quartic functions

We now test our algorithms. In a manner similar to that of Byrd et al. [6], we performed a set of controlled experiments. We applied our algorithms to the authors' set of 28 test problems which belong to a class of quartic functions (seven of them are quadratics). These tests are defined for $n = 100$ variables and consist of various difficulties in which the standard $\mathrm{L}_m D_1$ method works well on some of the tests and suffers on the others. Another feature which is reported in [6] is that these tests show

some improvement as the number of updates increases depending on certain difference vector pairs.

We will compare the methods for some values of the fixed numbers, for relative performance. We observed that both the number of line searches and the number of gradient evaluations required to solve the problems by the methods were slightly smaller than the number of function evaluations. Thus, we turn our attention to the latter number. A similar conclusion could be reported for the other two numbers.

Since our results were obtained on a large number of test cases, we will therefore present and discuss summaries and some average ratios extracted from these results. We briefly discuss the measure we constructed when designing our methods, which influences our treatment of the results of this paper. It is based on comparing the performance of several methods with that of $\mathrm{L}_m D_1$ for some values of m, in the following way. For $i = 1, 2, \ldots, 28$, let a_i and b_i be the number of function evaluations required by two methods (say, M1 and M2, respectively) to solve problem i,

$$r_i = \begin{cases} a_i/b_i & \text{if } a_i \leqslant b_i, \\ 2 - b_i/a_i & \text{otherwise,} \end{cases} \tag{23}$$

and r the average of these ratios. Then our average number of f calls required by M1 versus M2 is defined by

$$A_f = \begin{cases} r & \text{if } r \leqslant 1, \\ 1/(2 - r) & \text{otherwise.} \end{cases} \tag{24}$$

This modified average measure is identical to the average of the ratios a_i/b_i if $a_i \leqslant b_i$ (or $a_i \geqslant b_i$) for all i. It avoids certain difficulties associated with the usual average. For example, if two tests were solved by two methods with $a_1/b_1 = 0.5$ and $a_2/b_2 = 2$, then the usual average is 1.25, but our measure gives $A_f = 1$ indicating a fair result for comparing both methods. For further details concerning this modified measure, see Al-Baali [4]. Although this measure is not the only one which can be defined, it is hoped that it provides a sufficient basis to make the testing discussed in this section of interest.

We now give a brief idea about the choice (15) compared to the standard one (13). These choices define the basic matrices for the $\mathrm{L}_m D_2$ and $\mathrm{L}_m D_1$ methods, respectively. We applied these methods, using $m = 2, 3, \ldots, 40$, to the 28 tests. For convenience, the averages A_f of f calls were calculated versus the $\mathrm{L}_{10} D_1$ method. Some of these average ratios are given in table 2, where the column headings correspond to the values of m. As expected, both $\mathrm{L}_m D_1$ and $\mathrm{L}_m D_2$ improve as m increases. We observe that the latter method performs better than the former one for all values of m. Thus we will discuss below the results obtained for the choice D_2, unless otherwise stated.

To study the performance of the new algorithms, we will discuss some experiments based on the values $m = 20$, 15, 10, 7, 5, 4 and 3 with those of $p \leqslant m$. Although some of these values are larger than the practical ones, it is hoped that they provide a sufficient basis to illustrate some behaviour of the methods. We observed that for most of these choices our methods are competitive with the $\mathrm{L}_m D_1$ method.

Table 2
Average number of f calls required versus $\mathrm{L}_{10}D_1$.

Method $\setminus m$	40	30	25	20	15	10	7	5	4	3
$\mathrm{L}_m D_1$	0.67	0.69	0.70	0.80	0.87	1	1.11	1.19	1.26	1.48
$\mathrm{L}_m D_2$	0.65	0.67	0.67	0.76	0.85	0.95	1.02	1.11	1.13	1.34

Table 3
Average number of f calls required by $\mathrm{L}_m\mathrm{L}_p D_2$ and $R_p\mathrm{L}_m D_2$ versus $\mathrm{L}_{10}D_1$.

$p \setminus m$	15		10		7		5		4		3	
0	0.85	–	0.95	–	1.02	–	1.11	–	1.13	–	1.34	–
1	0.78	–	0.92	–	1.00	–	1.06	–	1.13	–	1.20	–
2	0.74	0.76	0.87	0.85	0.96	0.95	1.03	1.02	1.06	1.08	1.13	1.16
3	0.69	0.74	0.82	0.83	0.94	0.93	0.98	1.00	1.01	1.02	1.13	1.11
4	0.74	0.73	0.82	0.82	0.93	0.92	0.96	0.94	0.99	1.00	–	–
5	0.73	0.71	0.82	0.79	0.92	0.90	0.98	0.97	–	–	–	–
7	0.72	0.70	0.81	0.79	0.92	0.88	–	–	–	–	–	–
10	0.73	0.69	0.81	0.77	–	–	–	–	–	–	–	–
15	0.71	0.69	–	–	–	–	–	–	–	–	–	–

In general, $\mathrm{L}_p\mathrm{L}_m D_2$ and $R_p\mathrm{L}_m D_2$ are respectively the least and most efficient type of methods. The latter method is slightly better than the other two remaining algorithms.

To give some idea about these experiments, some results are summarized in table 3. The heading and the first column give some values of m and p, respectively. The results consist of the average number of f calls A_f required by $\mathrm{L}_m\mathrm{L}_p D_2$ (on the left) and $R_p\mathrm{L}_m D_2$ (on the right) versus $\mathrm{L}_{10}D_1$. The results for the choice $m = 20$ are not reported here, but they lie in $[0.67, 0.71]$ and $[0.64, 0.66]$ for both class of methods with $p \leqslant 6$ and $p \geqslant 7$, respectively (for further details see Al-Baali [2]). Note that the values of A_f required by the other two remaining methods $\mathrm{L}_m\mathrm{R}_p D_2$ and $\mathrm{L}_p\mathrm{L}_m D_2$ are usually close to those required by $\mathrm{L}_m\mathrm{L}_p D_2$.

It is clear that for each value of m the performance of the methods improves as p increases. An examination of these results shows the following. For most values of m and p, our three efficient methods perform slightly better than the $\mathrm{L}_{m+p}D_1$ method which depends on $m + p$ stored pairs and usually improves the performance of the $\mathrm{L}_m D_1$ method (substantially for $p \geqslant 2$).

For example, the average $A_f \approx 1$ appearing in table 3 for algorithms with 8 updates per each iteration sufficiently large shows that the performance of the $\mathrm{L}_4\mathrm{L}_4 D_2$, $\mathrm{L}_5\mathrm{L}_3 D_2$ and $\mathrm{L}_7\mathrm{L}_1 D_2$ methods is similar to that of $\mathrm{L}_{10}D_1$, which is usually better than that of $\mathrm{L}_8 D_1$.

Although $R_p\mathrm{L}_m D_2$ performs slightly better than $\mathrm{L}_m\mathrm{L}_p D_2$, the latter method is also desirable in practice, because it employs a number of updates smaller (substantially for large p) than that employed by the former (see table 1 for estimating the number of updates). Thus, in general, the $\mathrm{L}_m\mathrm{L}_p D_2$ method is economic in terms of not only

the number of function evaluations required to solve the tests, but also in terms of the number of updates.

It is worth mentioning that in another experiment which we will consider in the next subsection, $R_p\mathrm{L}_m D_2$ seems to be a little less efficient than $\mathrm{L}_m\mathrm{R}_p D_1$ for some values of m and p.

4.2. Applications to general functions

We now compare the performance of our algorithms with the standard L-BFGS method (denoted by $\mathrm{L}_m D_1$) on a set of 27 general test problems, most of which were used by Nash and Nocedal [13]. Table 4 presents names, abbreviated names and references of the problem set. The two tests EX-ROSBRKI and EX-POW-SRI are defined below. The CH-FRDRTH function is designed as CH-ROSBRK, though it is referred to as extended Freudenstein and Roth function by Toint [19]. The Q12 and Q20 functions belong to the set of 28 tests of Byrd et al. [6]. We have chosen them

Table 4
List of test problems.

Abbreviated name	Function's name	Reference
EX-ROSBRK	Extended Rosenbrock	Moré et al. [12]
EX-ROSBRKI	Extended Rosenbrock I	
EX-FRDRTH	Extended Freudenstein and Roth	Moré et al. [12]
CH-FRDRTH	Extended Freudenstein and Roth	Toint [19]
EX-POW-SR	Extended Powell Singular	Moré et al. [12]
EX-POW-SRI	Extended Powell Singular I	
VAR-DIM	Variably Dimensioned	Moré et al. [12]
PENALTYI	Penalty I	Moré et al. [12]
EX-ENGVL1	Extended ENGVL1	Toint [19]
WR-EX-WOOD	Wrong Extended Wood	Toint [19]
TRIGS	Trigonometric of Spedicato	Moré et al. [12]
TRIDB	Tridiagonal of Broyden	Moré et al. [12]
TRIDT	TRID (tridiagonal)	Toint [19]
SP-MXSQRT	Sparse Matrix Square Root	Liu and Nocedal [11]
LMS	Linear Minimum Surface	Toint [19]
TRIGT	TRIG (trigonometric)	Toint [19]
VAR(-3.4)	VAR(λ) (variational)	Toint [18]
MXSQRT1	Matrix Square Root 1	Liu and Nocedal [11]
MXSQRT2	Matrix Square Root 2	Liu and Nocedal [11]
Q12	Quartic type I ($\varepsilon = 0.09, \sigma = 0.18$)	Byrd et al. [6]
Q20	Quartic type II ($\varepsilon = 0.09, \sigma = 0.18$)	Byrd et al. [6]
CH-ROSBRK	Chained Rosenbrock	Fletcher [9]
PE	Potential Energy	Siegel [17]
BVP	Boundary Value Problem	Toint [19]
QOR	Quadratic Operations Research	Toint [18]
GOR	Generalized QOR	Toint [18]
PSP	Pseudo Penalty	Toint [18]

because the performance of the $L_{29}D_1$ method compared to certain methods proposed in that paper seems to work well on the former test and suffers on the latter one.

For the test problems EX-ROSBRK and EX-POW-SR, the standard starting points were used. These tests have the property that the components of all gradients generated during the optimization calculation satisfy the conditions

$$g_{i+2}^{(k)} = g_i^{(k)} \quad \text{and} \quad g_{i+4}^{(k)} = g_i^{(k)}, \tag{25}$$

respectively. Thus increasing the number of variables does not increase the number of function and gradient evaluations required by certain methods to solve these problems (for details, see Siegel [17], for instance). To avoid this occurrence, we replace the initial standard point, say $\bar{x}^{(1)}$, by

$$x_i^{(1)} = \frac{i}{i+1}\bar{x}_i^{(1)}, \quad i = 1, \ldots, n - \bar{n}, \tag{26}$$

where $\bar{n} = 2$ and 4 (the tests referred to as EX-ROSBRKI and EX-POW-SRI, respectively). Note that we applied certain methods to the EX-FRDRTH, VAR-DIM and PENALTYI problems using a choice similar to (26). Because the results were slightly different from that obtained for the standard initial point, we maintain the standard choice.

The number of variables n used for the runs ranges from 50 to 1000 (for details, see the second column of table 5).

It is worth noting that we have considered other test problems with $n \leqslant 10^4$. But we do not include them in our set because we observed that the above tests seem to give a reasonable idea about the behaviour of the limited memory methods.

We applied our algorithms, using practical choices of m ranging from 3 to 7 and values of $p \leqslant m$, to the above set of 27 tests. We observed that the average number of f calls required by these methods for $m \geqslant 5$ is similar to that given in table 3, and is a little better for $m \leqslant 4$. Most of the methods improve substantially over the corresponding $L_m D_1$ method, and some of them perform a little better than the $L_{10}D_1$ method. The most efficient methods for $m = 3, 4, \ldots, 7$ are, respectively, $R_3L_3D_2$, $L_4R_3D_1$, $L_5R_4D_2$, $L_6R_6D_2$ and $R_4L_7D_1$ (note that at each iteration sufficiently large, the number of updates employed by these methods are 8, 10, 15, 27 and 17, respectively).

Because employing a large number of updates is time-consuming, it is desirable to use the less efficient methods $L_m R_3 D_1$, for $m = 5, 6$ and 7, rather than the latter three methods (the number of updates become 11, 12 and 13, respectively). Similarly, we may use $L_3R_2D_1$ instead of $L_3R_3D_1$. The latter method performs slightly better (but requires 3 updates more) than the former one. It is worth noting that the performance of $L_4R_3D_1$ is slightly better than that of $L_4R_3D_2$, though both of them define identical steps whenever $k \geqslant 10$.

To illustrate the performance of the winner $L_4R_3D_1$ method, some results are presented in table 5. For comparison, we include the results required by the L_4D_1 and $L_{10}D_1$ methods. The results consist of the number of function and gradient evaluations

Table 5
Number of function and gradient evaluations.

Problem	n	L_4D_1		$\mathrm{L}_{10}D_1$		$\mathrm{L}_4R_3D_1$	
EX-ROSBRK	10^3	54	47	51	46	46	42
EX-ROSBRKI	10^3	326	303	491	469	183	166
EX-FRDRTH	10^3	22	21	23	23	19	15
CH-FRDRTH	10^3	82	51	52	42	41	35
EX-POW-SR	10^3	76	69	80	76	71	64
EX-POW-SRI	10^3	928	889	1270	1232	592	554
VAR-DIM	10^3	53	53	53	53	53	53
PENALTYI	10^3	75	72	77	75	75	71
EX-ENGVL1	10^3	25	23	26	23	25	23
WR-EX-WOOD	10^3	63	56	44	35	64*	53
TRIGS	10^3	89	85	84	78	73	71
TRIDB	10^3	40	36	40	35	36	34
TRIDT	10^3	491	480	449	446	356	352
SP-MXSQRT	10^3	212	208	205	200	186	184
LMS	961	231	228	258	251	217	214
TRIGT	100	161	159	141	138	142	140
VAR(-3.4)	100	871	863	570	564	602	597
MXSQRT1	100	527	523	438	432	428	426
MXSQRT2	100	541	533	510	505	435	432
Q12	100	878	853	730	715	907	890
Q20	100	738	703	562	528	467	446
CH-ROSBRK	100	570	560	584	569	541	537
PE	100	1788	1769	1604	1583	830	818
BVP	100	4189	4149	5948	5932	3629	3628
QOR	50	50	47	45	40	49	47
GOR	50	155	151	129	123	135	133
PSP	50	167	152	124	114	143	133

* Different local solution.

(left and right, respectively) required by these methods to solve each problem in the set. The asterisk indicates that the method terminates with a solution different from that obtained by the other methods given in the table.

We observe that the performance of the $\mathrm{L}_4R_3D_1$ method is substantially better than that of L_4D_1 in most cases. In general, this method is a little better than $\mathrm{L}_{10}D_1$. We note that the number of f calls required to solve some problems by $\mathrm{L}_4R_3D_1$ is equal to approximately 50% of the number required by L_4D_1 (e.g., CH-FRDRTH). This result is also observed even when the performance of the latter method is better than that of $\mathrm{L}_{10}D_1$ (e.g., EX-ROSBRKI). The opposite is observed on the Q12 test; $\mathrm{L}_{10}D_1$ and L_4D_1 perform better than $\mathrm{L}_4R_3D_1$. However, cases like this rarely occurred. Indeed we observed the desirable result that the number of function/gradient evaluations required to solve the Q12 test by $\mathrm{L}_4\mathrm{L}_3D_1$, $\mathrm{L}_3\mathrm{L}_4D_1$ and $R_3\mathrm{L}_4D_1$ (which are defined for $m = 4$) are 764/740, 795/781 and 768/747, respectively. These numbers are slightly greater than 730/715 which is required by the $\mathrm{L}_{10}D_1$ method (defined for

$m = 10$). Al-Baali [2] reported some improvement of other new methods on the Q12 test.

5. Concluding remarks

This paper proposes algorithms based on employing certain extra updates for the L-BFGS method without increasing the number of stored vector pairs. The experiments discussed in section 4 show that the new algorithms are competitive with the L-BFGS method and recommend the $\mathrm{L}_4R_3D_1$ method. For smaller storage, we suggest using $\mathrm{L}_3R_2D_1$ or $\mathrm{L}_2R_2D_1$ if necessary. If the problem is ill-conditioned or the value of m is large (say, $m \geqslant 7$), the $\mathrm{L}_m\mathrm{L}_{m-1}D_2$ method seems desirable.

An experiment worth mentioning is that we repeated the run described in section 4, except that some p extra updates were employed only at certain iterations depending on some heuristic argument. We observed that the new algorithms still improve over the L-BFGS method. We are thus concerned to keep at a minimum the number of times the extra updates are employed because they are time-consuming. Therefore, a future research subject is to seek a criterion to indicate whether to employ some extra updates. Then the total number of updates required to solve a problem could be reduced substantially.

Acknowledgements

I would like to express my gratitude to Lucio Grandinetti for providing an Italian university grant which partially supported this research. I would also like to thank Roger Fletcher and Jorge Nocedal for providing their routines which implement the limited memory BFGS method. Thanks are also due to Jorge for providing the implementation of his set of test problems. I am thankful to Philippe Toint for useful discussions and to the two anonymous referees for valuable comments made on the earlier draft of this paper.

References

[1] M. Al-Baali, New initial Hessian approximations for the limited memory BFGS method for large scale optimization, J. Fac. Sci., UAE University 14 (1995) 167–175.
[2] M. Al-Baali, Improved limited memory BFGS methods for large scale optimization, Technical Report DOMAS 97/1, Dept. of Mathematics and Statistics, Sultan Qaboos University, Oman (1997).
[3] M. Al-Baali, Extra updates for the BFGS method, Technical Report DOMAS 98/1, Dept. of Mathematics and Statistics, Sultan Qaboos University, Oman (1998).
[4] M. Al-Baali, Numerical experience with a class of self-scaling quasi-Newton algorithms, J. Optim. Theory Appl. 96 (1998) 533–553.
[5] B.M. Averick and J.J. Moré, Evaluation of large-scale optimization problems on vector and parallel architectures, SIAM J. Optim. 4 (1994) 708–721.

[6] R.H. Byrd, J. Nocedal and C. Zhu, Towards a discrete Newton method with memory for large scale optimization, Technical Report OTC 95/01, Optimization Technology Center, Argonne National Laboratory and Northwestern University, USA (1995).
[7] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. (Wiley, Chichester, UK, 1987).
[8] R. Fletcher, Low storage methods for unconstrained optimization, in: *Computational Solution of Nonlinear Systems of Equations*, eds. E.L. Allgower and K. George, Lectures in Applied Mathematics, Vol. 26 (Amer. Math. Soc., Providence, RI, 1990).
[9] R. Fletcher, An optimal positive definite update for sparse Hessian matrices, SIAM J. Optim. 5 (1995) 192–218.
[10] J.C. Gilbert and C. Lemaréchal, Some numerical experiments with variable storage quasi-Newton algorithms, Math. Programming 45 (1989) 407–436.
[11] D.C. Liu and J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Programming 45 (1989) 503–528.
[12] J.J. Moré, B.S. Garbow and K.E. Hillstrom, Testing unconstrained optimization software, ACM Trans. Math. Software 7 (1981) 17–41.
[13] S.G. Nash and J. Nocedal, A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization, SIAM J. Optim. 1 (1991) 358–372.
[14] J. Nocedal, Updating quasi-Newton matrices with limited storage, Math. Comp. 35 (1980) 773–782.
[15] S.S. Oren and D.G. Luenberger, Self-scaling variable metric (SSVM) algorithms, part I: Criteria and sufficient conditions for scaling a class of algorithms, Managm. Sci. 20 (1974) 845–862.
[16] D.F. Shanno and K.H. Phua, Matrix conditioning and nonlinear optimization, Math. Programming 14 (1978) 149–160.
[17] D. Siegel, Implementing and modifying Broyden class updates for large scale optimization, Technical Report NA12, Dept. of Applied Mathematics and Theoretical Physics, Cambridge University, England (1992).
[18] P.L. Toint, Some numerical results using a sparse matrix updating formula in unconstrained optimization, Math. Comp. 32 (1978) 839–851.
[19] P.L. Toint, Test problems for partially separable optimization and results for the routine PSPMIN, Report No. 83/4, Dept. of Mathematics, University of Namur, Belgium (1983).
[20] X. Zou, I.M. Navon, M. Berger, K.H. Phua, T. Schlick and F.X. Le Dimet, Numerical experience with limited-memory quasi-Newton and truncated Newton methods, SIAM J. Optim. 3 (1993) 582–608.