

A Numerical Study of Limited Memory BFGS Methods

José Luis Morales *

Abstract

The application of quasi-Newton methods is widespread in numerical optimization. Independently of the application, the techniques used to update the BFGS matrices seem to play an important role in the performance of the overall method. In this paper we address precisely this issue. We compare two implementations of the limited memory BFGS method for large-scale unconstrained problems. They differ in the updating technique and the choice of initial matrix. L-BFGS performs continuous updating, whereas SNOPT uses a restarted limited memory strategy. Our study shows that continuous updating techniques are more effective, particularly for large problems.

Keywords: quasi-Newton methods, BFGS formula, limited memory method.

1 Introduction

Quasi-Newton methods are very useful tools for solving unconstrained optimization problems, see for example [1], [2], [3]. These methods have proved to be efficient, robust, and relatively inexpensive in terms of computation. Their excellent properties have strongly motivated their use in other areas of numerical optimization. For example, certain practical realizations of the SQP algorithm make use of a limited memory BFGS method to maintain a positive definite approximation to the full Hessian of some augmented Lagrangian function. In practice there exist several ways to implement this method. We consider, as the main variations, different updating techniques, and different choices of the initial matrix. We argue that they may have a significant impact on the performance of the resulting method. In this paper we address precisely this issue. In our study we compare two implementations of the limited memory method: a) SNOPT, developed by Gill, Murray and Saunders [4]; b) L-BFGS as described by Nocedal [5]. Even though SNOPT is mainly intended for solving constrained problems, it provides us with a state of the art environment suitable for our comparison.

The organization of the paper is as follows. In section 2 we briefly review the BFGS method and set the notation. We also describe the main differences between L-BFGS and SNOPT. In section 3 we present our numerical study. The conclusions of our numerical study appear in section 4.

2 Limited Memory BFGS Methods

We start by formulating the BFGS method for solving the unconstrained optimization problem

$$\text{minimize} \quad f(x), \quad f: \mathbf{R}^n \rightarrow \mathbf{R}, \quad (1)$$

*ECE Department Northwestern University, Evanston IL 60208, USA. morales@ece.nwu.edu
www.ece.nwu.edu/~morales. On leave from Departamento de Matemáticas, Instituto Tecnológico Autónomo de México, Río Hondo 1, Col Tizapán San Angel, México D.F. CP 01000, México. jmorales@gauss.rhon.itam.mx. This author was supported by CONACyT grant 25710-A, the Fulbright Commission, National Science Foundation grant CCR 9907818, and by Asociación Mexicana de Cultura A.C.

where f is twice continuously differentiable function of a (possibly) large number of variables n . The BFGS method generates a sequence of iterates $\{x_k\}$ according to the following

Algorithm BFGS

1. Given x_0 , an initial approximation, and H_0 a positive definite approximation to the inverse of the Hessian at x_0 ,
2. **for** $k = 0$ until convergence **do**
 - compute the search direction

$$p_k = -H_k \nabla f(x_k) \tag{2}$$

- compute the step size by approximately solving the subproblem

$$\alpha_k = \arg \min_{\alpha > 0} f(x_k + \alpha p_k) \tag{3}$$

- compute the new iterate $x_{k+1} = x_k + \alpha_k p_k$
- compute H_{k+1} by updating H_k ,

end

where $\nabla f(x_k)$ stands for the gradient of f at the point x_k . The subproblem (3) is solved with a line search procedure that ensures that a set of sufficient decrease conditions are satisfied at the new point x_{k+1} . Typically the strong Wolfe conditions are used; these are,

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k, \tag{4}$$

$$|\nabla f(x_{k+1})^T p_k| \leq c_2 |\nabla f(x_k)^T p_k|, \tag{5}$$

$$0 < c_1 < c_2 < 1.$$

Finally, the matrix H_{k+1} is computed by updating H_k with the BFGS formula as follows

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \tag{6}$$

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k), \quad \rho_k = \frac{1}{y_k^T s_k}. \tag{7}$$

We now show that the updating pair (s_k, y_k) satisfies the curvature condition

$$s_k^T y_k > 0, \tag{8}$$

provided that the Wolfe conditions hold at the new point x_{k+1} . From the definition of y_k and the second condition (5) we have that

$$y_k^T s_k \geq (c_2 - 1) \alpha_k \nabla f(x_k)^T p_k,$$

where the right hand side of this inequality is always positive provided that p_k is a descent direction. Therefore, the BFGS formula ensures that the updated matrix H_{k+1} is positive definite.

2.1 L-BFGS

In the setting of large scale-optimization, the BFGS approach is not affordable due to memory constraints. The L-BFGS variant overcomes this difficulty by approximating the product in (2) in terms of the most recently computed m pairs $\{s_i, y_i\}$. When the $m + 1$ pair is computed, the oldest pair is discarded and its location in memory taken by the new pair.

An initial matrix \bar{H} that is updated using m pairs of the form $\{s_i, y_i\}_{i=1}^m$ is denoted as $\bar{H}(m)$. This approximation allows the computation of the product $\bar{H}(m)\nabla f(x_k)$ in $4mn + O(m)$ floating point operations; see Nocedal [5]. The memory requirements are $2mn + O(m)$ locations to hold the m pairs of vectors (s, y) . In L-BFGS the initial matrix \bar{H} is chosen as the identity matrix scaled by the quantity

$$\gamma_k = \frac{s_k^T y_k}{y_k^T y_k}. \quad (9)$$

2.2 SNOPT

The release 5.3-4 of SNOPT is an SQP code that makes use of a limited memory BFGS method to maintain an approximation to the full Hessian of a certain augmented Lagrangian function; see the details in [4]. Progress towards the solution is assessed by the use of a merit function. For unconstrained problems the augmented Lagrangian and the merit function reduce to f .

Instead of approximating the *inverse* of the Hessian, SNOPT approximates the Hessian itself. Therefore, the updating process is carried out by means of the formula

$$B_{k+1} = B_k + \theta_k y_k y_k^T - \phi_k q_k q_k^T, \quad (10)$$

where $q_k = B_k s_k$, $\theta_k = 1/s_k^T y_k$ and $\phi_k = 1/q_k^T s_k$. The vectors s_k and y_k are defined as in (7). The difficulties in memory and computational work associated with large-scale problems are avoided by means of a restarted variant of the procedure used in L-BFGS. This variant can be described as follows: a) a maximum of m pairs are kept in memory at any time; b) when the memory is full, the diagonal elements of the limited memory matrix defined by the current m pairs are stored into a diagonal matrix, and the m pairs are discarded; c) the initial matrix used to start a new cycle of m iterations is precisely this diagonal matrix. The updating formula (10) is applied directly for problems with a small number of nonlinear variables. In both cases a descent direction p_k is computed by solving (2) by means of a Cholesky factorization of B_k .

Since, in the general (constrained) case, the line search is performed on a *merit function*, there is no way to guarantee that the pair (s_k, y_k) will satisfy the curvature condition (8). Therefore, (s_k, y_k) is subject to the following test

$$s_k^T y_k \geq \sigma_k, \quad \sigma_k = \alpha_k (1 - \eta) p_k^T B_k p_k, \quad (11)$$

where η is a constant in the interval $(0, 1)$. If the pair fails the test (11) then the update is skipped.

3 Numerical Results

We now compare the performance of SNOPT and L-BFGS on a set of 130 unconstrained problems from the CUTE collection [6]. First we establish the main differences in the codes, then we discuss our experiments.

SNOPT uses the *pure* BFGS method (7) for problems in which the number of nonlinear variables is less than 75. If the number of nonlinear variables exceeds 75 then SNOPT makes use of the restarted procedure described in section 2. The restarts are applied every $m = 20$ iterations. L-BFGS uses continuous updating with $m = 20$, independently of the number of variables.

The line search procedure is slightly different in both implementations. SNOPT makes use of the following strategy: if $f(x_k + \alpha p_k) < f(x_k)$, and both values of f are distinguishable from a small tolerance, then it applies the following termination test based on the modified second strong Wolfe condition (5)

$$|\nabla f(x_{k+1})^T p_k| \leq (c_2 - c_1) |c_2 \nabla f(x_k)^T p_k|.$$

L-BFGS uses the Moré and Thuente implementation [7] of the strong Wolfe conditions (4-5). Both procedures use the same values of the parameters: $c_1 = 10^{-4}$, $c_2 = 0.9$.

Since the difference in the line search procedure could impact the results of our study, we conducted the following experiment: a) the line search used in SNOPT was implemented in L-BFGS; b) the two versions of L-BFGS were run on the set of problems. We obtained very similar results with both versions.

SNOPT skips updates that do not satisfy the sufficient curvature condition (11). We verified the number of times that updates were skipped. Just one update in one problem was skipped. SNOPT failed after skipping the update.

In our final comparison we have defined two subsets of problems. The set \mathcal{S}_1 is formed with 68 small problems whose dimension does not exceed 75. The set \mathcal{S}_2 contains 48 large scale problems. Since failures in the codes can be attributed to several causes and also may be difficult to explain, we included in our comparison only those problems in which both codes are successful.

All numerical experimentation was performed on an UltraSPARC 5 Sun Workstation with 384 MB of RAM memory, and machine precision of approximately $.222 \dots \times 10^{-16}$. The codes are written in Fortran and compiled with the `-O` option. Both codes use the stopping condition

$$\frac{\|\nabla f(x_k)\|_1}{\sqrt{n}} \leq TOL, \quad TOL = 1.0 \times 10^{-6}. \quad (12)$$

A limit of 2000 iterations was also imposed.

We now report on the numerical results. Our main indicator of performance is the relative number of function/gradient evaluations (fg), of methods A and B as measured by

$$r_{AB}^i = -\log_2[\text{fg}_A^i]/[\text{fg}_B^i],$$

where i stands for the i -th problem. The sign of r_{AB}^i indicates the winner (all cases in which A wins have a positive r_{AB}^i). The number of times by which the winner outperforms the loser is $2^{|r_{AB}^i|}$. Sometimes we will refer to this number as the *outperforming factor*. In our experiments A stands for L-BFGS, and B for SNOPT.

In Figure 1 we display the values of r_{AB}^i for the set of small problems. The name and corresponding dimension of each problem appear in Table 1, where problems have been placed in decreasing order with respect to their values of $|r_{AB}^i|$. Table 1 must be read row by row. Results for large problems are reported in Figure 2 and Table 2.

We observe that L-BFGS outperforms SNOPT in the vast majority of the low dimensional problems. This seems surprising due to the fact that SNOPT uses the BFGS method in its *pure* form. This observation confirms previous experience [8]. We explain this difference in behavior by recalling that L-BFGS keeps the most recent information on the curvature of the problem. On the other hand, BFGS updates old information, which may be irrelevant in most nonlinear problems. Observe that the differences are substantial: the outperforming factor is greater than 2 in approximately 10 problems.

For large scale problems the difference in performance is much more pronounced. L-BFGS outperforms SNOPT, with larger outperforming factors than those observed in the small scale problems, in the majority of problems. We now observe outperforming factors greater than 2 in approximately half of the problems in the set.

Table 1: Statistics for the small dimension problems.

Problem	n	Problem	n	Problem	n	Problem	n	Problem	n
BIGGS3	6	QUARTC	25	SROSENBR	2	PFIT1LS	3	AIRCRAFTB	8
WOODS	4	BROWNDEN	4	ZANGWIL2	2	HILBERTA	2	DENSCHND	3
LMINSURF	16	CLIFF	2	DENSCHNF	2	HILBERTB	5	BIGGS5	6
HAIRY	2	EXPFIT	2	DENSCHNE	3	ENGVAL2	3	LIARWHD	36
BROWNAL	10	MEXHAT	2	PFIT4LS	3	DIXMAANC	15	BOX3	3
CUBE	2	POWELLSG	4	BRKMCC	2	FLETCHCR	10	BARD	3
BOX2	3	MOREBV	10	DIXMAANI	15	DIXMAANA	15	DENSCHNA	2
BIGGS6	6	GULF	3	DIXMAANE	15	SISSER	2	WATSON	12
VARDIM	10	ALLINITU	4	MSQRTBLS	9	DENSCHNB	2	EDENSCH	36
ENGVAL1	2	PFIT3LS	3	JENSMP	2	TRIDIA	30	OSBORNEB	11
DIXMAANK	15	DIXMAANH	15	HELIX	3	PFIT2LS	3	DENSCHNC	2
DIXMAANL	15	MSQRTALS	4	OSBORNEA	5	DIXMAANF	15	DIXMAANG	15
KOWOSB	4	DQDRTIC	10	SINQUAD	5	SINEVAL	2		

Table 2: Statistics for the large dimension problems.

Problem	n	Problem	n	Problem	n	Problem	n
ENGVAL1	1000	EG2	1000	SENSORS	100	SINQUAD	1000
SPARSQR	1000	DIXMAANL	300	NONDQUAR	1000	PENALTY1	1000
DIXMAANI	300	WOODS	1000	BDQRTIC	1000	DIXMAANK	300
TOINTGSS	1000	DIXMAANC	300	DIXMAANB	300	VARDIM	100
POWELLSG	1000	DIXMAAND	300	ARWHEAD	1000	DIXMAANJ	300
DQRTIC	1000	QUARTC	1000	ARGLINA	100	SPMSRTL	1000
NCB20	110	NONDIA	1000	FMINSURF	961	VAREIGVL	1000
PENALTY2	100	DIXMAANE	300	BRYBND	1000	EIGENALS	110
MOREBV	1000	SCHMVETT	1000	POWER	1000	LIARWHD	1000
DIXMAANA	300	TRIDIA	1000	SROSENBR	1000	COSINE	1000
DIXMAANG	300	GENROSE	500	DIXMAANF	300	FLETCHCR	100
TQUARTIC	1000	DIXMAANH	300	CRAGGLVY	1000	DQDRTIC	1000

Figure 1. Relative performance of L-BFGS and SNOPT for problems in set \mathcal{S}_1 .

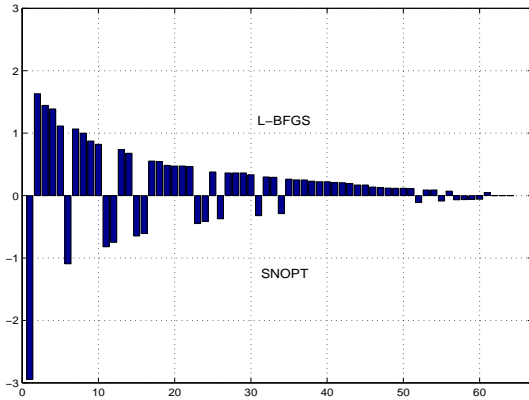
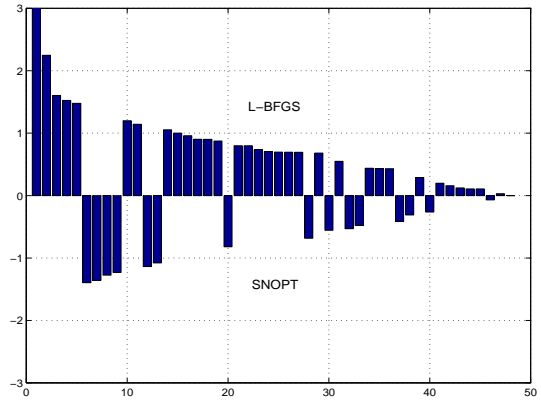


Figure 2. Relative performance of L-BFGS and SNOPT for problems in set \mathcal{S}_2 .



4 Conclusions

We have presented a numerical study of two quasi-Newton methods for unconstrained optimization. They differ mainly in their strategy for updating the BFGS matrices, and in the choice of the initial matrix. The effect of other computational differences, the line search procedure and the test of positive curvature, has been assessed and then discarded by means of appropriate experimentation.

Our numerical study indicates that storing the most recently computed curvature (L-BFGS), independently of the number of variables, is a flexible and reliable scheme. On the other hand, accumulating old information by keeping full matrices as in the case of small problems (or their diagonals as in the large scale case), is less effective in the problems studied. The difference in performance is significantly more pronounced in the large scale case. It is not clear that the same conclusion would be valid for the constrained case, but this is certainly an issue that deserves further research.

Acknowledgments

The author is indebted to Jorge Nocedal for his help and encouragement during this project. The author also wishes to thank an anonymous referee for comments and remarks that significantly improved the paper.

References

- [1] J.E. Dennis, Jr. and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, N.J., Prentice-Hall, (1983).
- [2] R. Fletcher, *Practical Methods of Optimization*, 2nd Edition, New York, John Wiley & Sons, (1987).
- [3] J. Nocedal and S.J. Wright, *Numerical Optimization* New York, Springer-Verlag, (1999).
- [4] P.E. Gill, W. Murray, and M.A. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, Report NA 97-2, Dept of Mathematics, University of California, San Diego.
- [5] J. Nocedal, Updating quasi-Newton matrices with limited storage, *Math. Comput.*, **35** 773–782(1980).
- [6] I. Bongartz, A.R. Conn, N.I.M. Gould, and Ph.L. Toint, CUTE: Constrained and Unconstrained Testing Environment, *ACM Trans. Math. Software*, **21** 123–160 (1995).
- [7] J.J. Moré and D.J. Thuente, Line search algorithms with guaranteed sufficient decrease, *ACM Trans. Math. Software*, **20** 286–307 (1994).
- [8] J. Nocedal, Personal communication.