NORTHWESTERN UNIVERSITY
Department of Electrical Engineering
and Computer Science

L-BFGS-B – FORTRAN SUBROUTINES FOR LARGE-SCALE BOUND CONSTRAINED OPTIMIZATION

by

Ciyou Zhu[1], *Richard H. Byrd*[2], *Peihuang Lu*[1] *and Jorge Nocedal*[1]

December 31, 1994

ABSTRACT

L-BFGS-B is a limited memory algorithm for solving large nonlinear optimization problems subject to simple bounds on the variables. It is intended for problems in which information on the Hessian matrix is difficult to obtain, or for large dense problems. L-BFGS-B can also be used for unconstrained problems, and in this case performs similarly to its predecessor, algorithm L-BFGS (Harwell routine VA15). The algorithm is implemented in Fortran 77.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization – *gradient methods*; G.4 [**Mathematics of Computing**]: Mathematical Software.

General Terms: Algorithms

Additional Key Words and Phrases: variable metric method, large scale optimization, nonlinear optimization, limited memory method.

[1] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston Il 60208. These authors were supported by National Science Foundation Grants CCR-9101359 and ASC-9213149, and by Department of Energy Grant DE-FG02-87ER25047-A004.
[2] Computer Science Department, University of Colorado at Boulder, Boulder Colorado 80309. This author was supported by NSF grant CCR-9101795, ARO grant DAAL 03-91-G-0151, and AFOSR grant AFOSR-90-0109.

1. Introduction.

The purpose of algorithm L-BFGS-B is to minimize a nonlinear function of n variables,

$$\min f(x)$$

subject to the simple bounds

$$l \leq x \leq u,$$

where the vectors l and u represent lower and upper bounds on the variables. Not all the variables need to have bounds; in fact the algorithm is also appropriate and efficient for solving unconstrained problems. The user must supply the gradient g, but knowledge about the Hessian matrix of f is not required. For this reason the algorithm can be useful for solving large problems in which the Hessian matrix is difficult to compute or is dense.

The algorithm is described in detail in [8], and proceeds roughly as follows. At each iteration a limited memory BFGS approximation to the Hessian is updated. This limited memory matrix is used to define a quadratic model of the objective function f. A search direction is then computed using a two-stage approach: first, the gradient projection method [15], [3], [18],[9] is used to identify a set of active variables, i.e. variables that will be held at their bounds; then the quadratic model is approximately minimized with respect to the free variables. The search direction is defined to be the vector leading from the current iterate to this approximate minimizer. Finally a line search is performed along the search direction using the subroutine described in [17]. A novel feature of the algorithm is that the limited memory BFGS matrices are represented in a compact form [7] that is efficient for bound constrained problems.

The user can control the amount of storage required by L-BFGS-B by selecting a parameter m that determines the number of BFGS corrections saved. The algorithm requires roughly $(12 + 2m)n$ storage locations, and since small values of m (say $3 \leq m \leq 20$) are recommended, it can be used to solve very large problems. The computational cost of one iteration of the algorithm is modest, ranging from $4mn + n$ multiplications when no bounds are active, to approximately m^2n multiplications when all variables are at their bounds.

If no bounds are active at the solution, it is appropriate to stop the iteration when the norm of the gradient g is sufficiently small. The corresponding quantity for the case when some bounds are active is the norm of the projected gradient, which we denote by $\|\text{proj } g\|$, and which is defined, for example, in [8]. Both the output of L-BFGS-B and its documentation, make reference to the projected gradient.

L-BFGS-B is an extension of the limited memory algorithm (L-BFGS) for unconstrained optimization described in [16] and implemented as Harwell routine VA15 [12]. The main improvement is the ability of L-BFGS-B to deal with bounds on the variables. Even though this requirement makes the new algorithm far more complex than its predecessor, the two codes perform similarly on unconstrained problems. Therefore L-BFGS-B could be considered to supersede L-BFGS – except for one fact that can be important in some applications: L-BFGS-B requires 8 more n-vectors of storage.

L-BFGS-B is, at present, the only limited memory quasi-Newton algorithm capable of handling bounds on the variables; other published codes [5], [6], [13], [16] are only able to solve unconstrained problems. We note also that the nonlinear conjugate gradient method [14], which is used for solving many large unconstrained problems, has not been adequately extended to handle bounds on the variables, and L-BFGS-B can be used in its place.

The advantages of L-BFGS-B are: (i) the code is easy to use, and the user need not supply information about the Hessian matrix or about the structure of the objective function; (ii) the storage requirements of the algorithm are modest and can be controlled by the user; (iii) the cost of the iteration is low, and is independent of the properties of the objective function. Due to this, L-BFGS-B is recommended for solving large problems in which the Hessian matrix is not sparse or is difficult to compute.

However L-BFGS-B suffers from the following drawbacks: (i) it is not rapidly convergent, and on difficult problems can take a large number of function evaluations to converge; (ii) on highly ill-conditioned problems the algorithm may fail to obtain high accuracy in the solution; (iii) the algorithm cannot make use of knowledge about the structure of the problem to accelerate convergence.

The code can be obtained by anonymous ftp to eecs.nwu.edu. After logging in go to the directory pub/lbfgs.

2. How to use the routine.

The simplest way to use the code is to modify one of the drivers provided in the package (see section 3). Most users will only need to make a few changes to the drivers to run their applications.

L-BFGS-B is written in FORTRAN 77, in double precision. The user is required to calculate the function value f and its gradient g. In order to allow the user complete control over these computations, reverse communication is used. The routine lbfgsb.f must be called repeatedly under the control of the variable **task**. The calling statement of L-BFGS-B is

call lbfgsb(n,m,x,l,u,nbd,f,g,factr,wa,iwa,task,iprint,isbmin,csave,lsave,isave,dsave)

Following is a description of all the parameters used in this call.

n is an INTEGER variable that must be set by the user to the number of variables. It is not altered by the routine.

m is an INTEGER variable that must be set by the user to the number of corrections used in the limited memory matrix. It is not altered by the routine. Values of **m** less than 3 are not recommended, and large values of **m** can result in excessive computing time. The range $3 \leq \mathbf{m} \leq 20$ is recommended.

x is a DOUBLE PRECISION array of length **n** . On initial entry it must be set by the user to the values of the initial estimate of the solution vector. Upon successful exit, it contains the values of the variables at the best point found.

l is a DOUBLE PRECISION array of length **n** that must be set by the user to the values of the lower bounds on the variables. If the i-th variable has no lower bound, **l**(i) need not be defined.

u is a DOUBLE PRECISION array of length **n** that must be set by the user to the values of the upper bounds on the variables. If the i-th variable has no upper bound, **u**(i) need not be defined.

nbd is an INTEGER array of dimension **n** that must be set by the user to the type of bounds imposed on the variables:

nbd(i) = 0 if **x**(i) is unbounded,
1 if **x**(i) has only a lower bound,

2 if **x**(i) has both lower and upper bounds

3 if **x**(i) has only an upper bound.

f is a DOUBLE PRECISION variable. If the routine lbfgsb.f returns with **task**(1:2)= 'FG', **f** must be set by the user to contain the value of the function f at the point **x**.

g is a DOUBLE PRECISION array of length **n** . If the routine lbfgsb.f returns with **task**(1:2)= 'FG', **g** must be set by the user to contain the components of the gradient g at the point **x**.

factr is a DOUBLE PRECISION variable that must be set by the user. It is a tolerance in the termination test for the algorithm. The iteration will stop when

$$(\mathbf{f}_k - \mathbf{f}_{k+1})/\max(|\mathbf{f}_{k+1}|, |\mathbf{f}_k|, 1) \leq \mathbf{factr}{*}\mathbf{epsmch} \quad (1)$$

where **epsmch** is the machine precision which is automatically generated by the code. Typical values for **factr** on a computer with 15 digits of accuracy in double precision are: **factr**=1.d+12 for low accuracy; **factr**=1.d+7 for moderate accuracy; **factr**=1.d+1 for extremely high accuracy. If the user sets **factr**=0, the test will stop the algorithm only if the objective function remains unchanged after one iteration.

wa is a DOUBLE PRECISION array of length $(2\mathbf{m}+4)\mathbf{n} + 12\mathbf{m}^2 + 12\mathbf{m}$ used as workspace. This array must not be altered by the user.

iwa is an INTEGER array of length 3**n** used as workspace. This array must not be altered by the user.

task is a CHARACTER string of length 60 that must be set to 'START' on initial entry. On a return with **task**(1:2)='FG' the user must evaluate the function **f** and gradient **g** at the returned value of **x**. On a return with **task**(1:5) ='NEW_X' an iteration of the algorithm has concluded, and **f** and **g** contain $f(\mathbf{x})$ and $g(\mathbf{x})$ respectively. The user can decide whether to continue or stop the iteration. When:

task(1:4)='CONV' the termination test in L-BFGS-B has been satisfied;

task(1:4)='ABNO' the routine has terminated abnormally without being able to satisfy the termination conditions; **x** contains the best approximation found; **f** and **g** contain $f(\mathbf{x})$ and $g(\mathbf{x})$ respectively;

task(1:5)='ERROR' the routine has detected an error in the input parameters.

On exit with **task** ='CONV', 'ABNO' or 'ERROR', the variable **task** contains additional information that the user can print. This array should not be altered by the user unless the user wants to stop the run for some reason. See driver2.f and driver3.f for detailed explanation on how to stop the run by assigning task(1:4)='STOP' in the driver.

iprint is an INTEGER variable that must be set by the user. It controls the frequency and type of output generated:

iprint=-1 no output is generated;

iprint=0 print only one line at the last iteration;

0<iprint<99 print also **f** and ||proj **g**|| every **iprint** iterations;

iprint=99 print details of every iteration except components of n-vectors;

iprint=100 print also the changes of active sets and the final n-vector **x**;

iprint=101 print details of every iteration including the changes of active sets and the n-vectors **x** and **g**.

When **iprint** > 0, the file iterate.dat will be created to summarize the iteration.

isbmin is an INTEGER variable that must be set by the user. The default setting is **isbmin** =1. This variable determines the type of method used for the solution of the inner subspace minimization subproblems,

isbmin = **1** the direct primal method will be used,

2 the dual method will be used,

3 the conjugate gradient method will be used.

csave is a CHARACTER working array of length 60. This array must not be altered by the user.

lsave is a LOGICAL working array of dimension 4. This array must not be altered by the user. On exit with **task** = 'NEW_X', the following information is available:

lsave(1) = .true. the initial **x** did not satisfy the bounds;

lsave(2) = .true. the problem contains bounds;

lsave(3) = .true. each variable has upper and lower bounds;

isave is an INTEGER working array of dimension 44. This array must not be altered by the user. On exit with **task** = 'NEW_X', it contains information that the user may want to access. For example,

isave(30) = the current iteration number;

isave(34) = the number of function and gradient evaluations performed so far.

See the subroutine lbfgsb.f for a description of other information contained in **isave**.

dsave is a DOUBLE PRECISION working array of dimension 29. This array must not be altered by the user. On exit with **task** = 'NEW_X', it contains information that the user may want to access. For example,

dsave(2) = value of **f** at the previous iteration;

dsave(13) = the infinity norm of the projected gradient of f at **x**.

See the subroutine lbfgsb.f for a description of other information contained in **dsave**.

3. The Drivers.

Several sample drivers have been prepared to facilitate the use of the code. They range from a simple driver using all the default settings, to some more sophisticated drivers that give the user more control over the execution of the code.

driver1.f is the simplest driver. It demonstrates how to solve a sample problem using all the default settings of the code. We recommend that every user of L-BFGS-B read this driver. It gives a good idea of how the code works, and at the end of driver1.f there is a detailed description of the parameters used in L-BFGS-B.

driver2.f is a more sophisticated driver. It illustrates various ways of terminating the run, and alternative ways of generating output. This driver is designed for users who need specially formatted output or for users who wish to have more control over the execution of the run.

driver3.f is a time-controlled driver. It shows how to terminate a run after some prescribed CPU time has elapsed, and how to print the desired information before exiting. When running very time-consuming applications the user may wish to impose a limit on CPU time. Terminating the run in this way, however, will not produce the final output of the run. This driver shows how to generate all desired output in this case.

driver4.f is an extensive driver. It runs the code on 67 test problems (33 bound constrained, 34 unconstrained) from the CUTE collection [4], each with three different subspace minimization methods. This driver is used for testing and profiling the code.

4. Termination and Error Messages.

The user can terminate execution at various stages of the algorithm by setting task(1:4)= 'STOP' at an appropriate place in the driver. This allows the user to determine a stopping condition based on such factors as projected gradient, number of function evaluations or time spent. Several possibilities are illustrated in driver2.f. In addition, the code may terminate because the built-in stopping test (1) has been met, because an input error has been detected, or because the code cannot make further progress.

The built-in stopping test (1) is controlled by the parameter **factr**; see section 2. It is designed to terminate the run when the change in the objective function **f** is sufficiently small. The test can be made more stringent by decreasing **factr**, and can be almost disabled by setting **factr**= 0. If the stopping test is satisfied, **task**(1:4) will contain the string 'CONV'. Note that (1) is scale-dependent due to the scalar 1 in the denominator: when **factr** > 0, if **f** is multiplied by a constant and the code is re-run, then termination may occur at a different solution point.

In addition to the stopping test (1), the algorithm has another built-in stopping test based on the projected gradient: if

$$\|\text{proj}\,\mathbf{g}\|_\infty = 0$$

the execution will terminate. This test is included because the algorithm can proceed only when the projected gradient is nonzero; otherwise a stationary point has already been reached. In this case the variable **task** contains the string 'CONVERGENCE: NORM OF PROJECTED GRADIENT = 0'.

It can occur sometimes that the line search cannot make any progress, as described in section 5. In this case the run is terminated, and **task** contains the string 'ABNORMAL TERMINATION IN LINE SEARCH'.

When the code has detected an error in the input, **task**(1:5) will contain the string 'ERROR'. In this case the user can print all the information contained in **task** (i.e **task**(1:60)), which will provide details of the error.

5. Implementation.

The algorithm implemented in L-BFGS-B is described in detail in [8]. However a few additions and modifications were made during the development of the code.

First we describe several devices for dealing with failures of the code and for trying to improve performance in the region were rounding errors begin to dominate the computation. The line search program of Moré and Thuente [17] is used to compute the steplength parameter. If the line search is unable to find a point with a sufficiently lower value of the objective after 20 evaluations of the objective function, we conclude that the current direction is not useful. In this case all correction vectors are discarded and the iteration is restarted along the steepest descent direction. If the line search fails along this steepest descent direction, the algorithm terminates as described in section 4. This type of failure will usually occur only if the user has specified high accuracy in the solution and L-BFGS is having difficulties meeting this accuracy. Our restarting strategy sometimes leads to successful termination in these difficult cases – but not always.

Similarly, if during the course of the iteration the L-BFGS matrix, or a related submatrix becomes singular or indefinite, all correction vectors are discarded and the iteration is restarted along the steepest descent direction. This device is also used if the search direction is not a descent direction (i.e. if $g^T d \geq 0$). We emphasize that all the difficulties just described occur only when rounding errors begin to dominate the computation.

Machine and Scale Dependencies.

L-BFGS-B computes the machine precision **epsmch** by means of the routine **dpmeps** from MINPACK-2 [2]. The machine precision **epsmch** is used only twice in the algorithm: in the stopping test (1) and in the skipping criterion for BFGS updating described below. These two computations are therefore machine-dependent. As explained in the previous section, the stopping test (1) can be controlled by means of the variable **factr**; the user may want to experiment with this variable. When **factr** is set equal to zero, this machine dependence is removed. The BFGS skipping test is necessary to guarantee the positive definiteness of the limited memory matrices when bounds on the variables are present. In L-BFGS-B the matrix update is skipped when

$$\frac{y_k^T s_k}{-g_k^T s_k} \leq \mathbf{epsmch},$$

where $y_k = g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$; (see [11]). This ensures that $y_k^T s_k$ is sufficiently positive. The user can determine how many times the BFGS update was skipped by printing the variable isave(26). Our numerical experience indicates that skipping occurs rarely.

Effort was taken to ensure that L-BFGS-B is as scale-invariant as possible. However complete scale-invariance was not possible to achieve; indeed the limited memory algorithm itself is not invariant to linear transformations in the variables. However, the algorithm is invariant with respect to scalar multiples of the variables and the objective function, and we have been able to maintain that invariance in the code with only a few exceptions. The main exception is the first iteration, where the step is quite dependent on scaling of the variables. In addition, as noted before, the test (1) is not invariant to scaling of the objective function when $\mathbf{factr} > 0$. However, since this test can easily be altered or essentially disabled by the user, this is not an important drawback.

A technical point on the step computation.

There is one significant difference between L-BFGS-B and the algorithm described in [8], but it occurs at a fairly low level and is of interest only to those readers wishing to understand the code in detail.

The implementation of the primal and dual approaches for subspace minimization was unified. With the notation used in section 5 of [8], it can be shown that the matrix

$$(I - \frac{1}{\theta} M W^T Z Z^T W)^{-1} M,$$

which appears in the primal direct method is identical to the matrix

$$(I + \theta \bar{M} \bar{W}^T A A^T \bar{W})^{-1} \bar{M}$$

of the dual method. Moreover, these matrices can be written as the inverse of

$$\begin{bmatrix} -D - \frac{1}{\theta} Y^T Z Z^T Y & L_A^T - R_Z^T \\ L_A^T - R_Z^T & \theta S^T A A^T S \end{bmatrix},$$

where L_A is the strict lower triangle of $S^T A A^T S$ and R_Z is the upper triangle of $S^T Z Z^T Y$. We have used this matrix in both the primal and dual approaches, and as a result the performance of the two methods is now similar. Although this matrix is not positive definite, it can be factorized symmetrically by using Cholesky factorization of the submatrices.

6. Numerical Results.

We now present results of L-BFGS-B on a set of test problems from the CUTE collection [4]. We tested only bound constrained problems with $n \geq 5$ and unconstrained problems with $n \geq 100$. As a benchmark we also present the results obtained by the SR1 and Exact Hessian options of the LANCELOT package [10]. LANCELOT was run using all its default options. All runs were performed on a Sparcstation-2 with 32Mb of main memory, and all runs were terminated when the norm of the projected gradient is less than 10^{-5}, i.e.

$$\|\text{proj } \mathbf{g}\|_\infty \leq 10^{-5}. \tag{2}$$

The meaning of some of the variables used in the tables is as follows.

nbnd: the number of active bounds at the solution of LANCELOT–SR1.

nfg: the total number of function or gradient evaluations.

nf: the total number of function evaluations. (In LANCELOT, the number of function evaluations may differ from the number of gradient evaluations.)

Tables 1.1 and 1.2 indicate that L-BFGS-B is a competitive code, which is remarkable since it does not use any specific knowledge of the objective function, as is the case in both versions of LANCELOT. Although, as is to be expected, L-BFGS-B used more function evaluations than LANCELOT, the CPU times were comparable, though with great variability. It is an interesting fact that L-BFGS-B is sometimes unable to reduce the projected gradient sufficiently to satisfy the stopping condition even though the function value obtained is very good. More specifically, in the runs marked by C1 in the tables, L-BFGS-B obtained at least as good function value (to five digits) as LANCELOT but the gradient did not meet the stopping condition. We do not interpret these as failures of the algorithm, but feel that this property of L-BFGS deserves further study.

Tables 1.3 and 1.4 show the effect of varying the number m of updates saved. Increasing m definitely improves the reliability of the algorithm. Although increasing m often reduces the number of function evaluations, this effect is not consistent, and it does cause an increase in CPU time in most cases. In Tables 1.5 and 1.6 we consider the dual and conjugate gradient approaches for subspace minimization. With the new implementation described in section 5, the computation time for the primal and dual approaches is quite similar. The conjugate gradient approach seems to require more time on problems where computational time is significant.

BOUND CONSTRAINED PROBLEMS

Problem	n	nbnd	L-BFGS-B m=5 (Primal)		L-BFGS-B m=17 (Primal)		LANCELOT SR1		LANCELOT Hessian	
			nfg	time	nfg	time	nf	time	nf	time
BDEXP	1000	0	15	2.31	16	3.50	27	13.74	11	6.54
BIGGS5	6	1	121	0.88	69	1.51	41	0.62	19	0.30
BQPGASIM	50	7	25	0.28	23	0.43	8	0.58	4	0.34
BQPGAUSS	2003	27	*F1	(7E–3)	*C1	(4E–4)	20	1957.59	9	1751.29
HATFLDC	25	0	23	0.19	23	0.41	5	0.11	5	0.19
HS110	50	50	2	0.02	2	0.02	2	0.17	2	0.16
HS45	5	5	11	0.03	11	0.01	3	0.05	3	0.03
JNLBRNGA	15625	5657	332	740.33	296	1133.88	24	1263.77	22	1502.96
JNLBRNGB	1024	516	424	62.73	426	125.17	6	7.21	6	5.56
LINVERSE	999	338	291	56.85	369	159.31	27	194.08	28	149.99
MAXLIKA	8	1	1665	88.38	158	10.27	98	24.33	9	2.24
MCCORMCK	1000	0	15	1.85	15	2.05	7	5.25	5	3.97
NONSCOMP	1000	2	45	6.79	60	17.24	9	4.70	9	4.43
OBSTCLAE	5625	2724	258	207.20	308	455.60	7	1442.00	6	1422.62
OBSTCLAL	1024	508	40	5.84	40	10.45	11	9.45	9	7.69
OBSTCLBL	1024	475	50	7.83	55	16.62	8	15.42	8	18.45
OBSTCLBM	15625	4309	146	353.04	138	573.84	7	1106.37	6	2017.70
OBSTCLBU	1024	475	44	6.57	41	11.48	9	16.10	8	8.45
PALMER1A	6	0	799	4.95	262	4.50	113	2.29	68	1.37
PALMER1E	8	0	*F1	(7E–2)	290	5.06	190	6.95	204	7.38
PALMER2A	6	0	518	3.67	182	4.12	180	3.05	157	2.60
PALMER2E	8	0	*F1	(2E–3)	291	6.98	268	8.01	113	3.89
PALMER3A	6	0	716	5.13	140	3.31	176	3.02	147	2.48
PALMER3E	8	0	*F1	(4E–4)	221	3.59	141	3.81	68	1.76
PALMER4A	6	0	483	3.30	128	2.82	98	1.54	48	0.80
PALMER4E	8	0	*F1	(3E–3)	172	2.89	206	4.78	67	1.95
PROBPENL	500	0	3	0.10	3	0.11	3	1.72	2	1.67
S368	100	29	21	16.84	21	16.93	37	91.24	8	21.14
TORSION1	1024	436	43	6.35	32	8.16	13	11.04	11	10.18
TORSION2	1024	436	61	10.08	55	18.33	10	12.31	5	12.69
TORSION3	1024	748	23	2.76	22	3.61	7	8.43	6	4.05
TORSION4	1024	748	49	5.87	43	8.32	7	6.73	6	4.85
TORSION6	14884	12316	362	707.22	360	1157.74	10	130.73	9	130.31

Table 1.1. Test results of L-BFGS-B, using the default option (primal method) for subspace minimization, and results of LANCELOT's SR1 and exact Hessian options, on bound constrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.
F1: Gradient stopping test (2) was not met but the final function value was greater than that obtained by LANCELOT SR1.

UNCONSTRAINED PROBLEMS

Problem	n	L-BFGS-B m=5 (Primal)		L-BFGS-B m=17 (Primal)		LANCELOT SR1		LANCELOT Hessian	
		nfg	time	nfg	time	nf	time	nf	time
ARWHEAD	1000	13	1.09	**C1	(2E–5)	5	4.66	6	4.79
BDQRTIC	100	101	1.28	47	1.29	11	1.06	12	1.07
BROYDN7D	1000	373	66.30	398	104.51	112	62.72	125	66.52
CRAGGLVY	1000	95	13.33	89	19.08	15	9.81	15	9.89
DIXMAANA	1500	12	1.34	13	1.66	8	8.71	6	7.96
DIXMAANB	1500	12	1.36	12	1.43	9	10.18	8	8.96
DIXMAANC	1500	14	1.61	14	1.85	10	8.91	12	11.28
DIXMAAND	1500	15	1.70	15	2.08	13	13.07	20	15.73
DIXMAANE	1500	188	24.28	169	41.07	14	13.01	7	8.74
DIXMAANF	1500	163	21.04	126	30.71	26	21.17	33	22.11
DIXMAANG	1500	158	20.38	127	30.94	32	24.78	25	18.05
DIXMAANH	1500	156	20.30	124	30.01	37	28.07	36	24.44
DIXMAANI	1500	1237	166.37	1066	273.08	11	11.56	8	9.30
DIXMAANK	1500	130	16.59	146	35.36	34	25.98	51	32.56
DIXMAANL	1500	134	16.93	120	28.04	105	67.67	50	35.88
DQDRTIC	1000	19	1.47	19	1.73	3	2.47	3	2.55
DQRTIC	500	43	1.46	43	2.96	34	6.13	34	6.11
EIGENALS	110	574	17.21	302	15.77	21	4.72	22	4.36
EIGENBLS	110	1116	33.36	1041	55.73	186	98.47	193	95.55
EIGENCLS	462	2900	563.81	2507	599.32	456	2010.40	543	2299.42
ENGVAL1	1000	23	2.02	20	2.38	8	6.28	8	6.03
FLETCHBV	100	**C1	(2E–0)	**C1	(9E–1)	*F2	(2E+4)	*F1	(3E+4)
FREUROTH	1000	**C1	(2E–5)	**C1	(1E–3)	11	7.53	11	7.27
GENROSE	500	1244	60.86	1315	116.82	590	103.92	586	99.79
MOREBV	1000	79	6.85	77	12.22	2	3.89	2	3.85
NONDIA	1000	23	1.79	23	2.56	C2	C2	30	12.54
NONDQUAR	100	1001	10.09	828	25.82	16	0.86	16	0.86
PENALTY1	1000	60	3.91	60	7.58	64	118.89	64	117.61
PENALTY3	100	**C1	(3E–3)	**C1	(3E–3)	100	436.12	**C1	(2E–4)
QUARTC	1000	47	3.10	47	5.86	36	12.74	36	12.68
SINQUAD	1000	183	17.17	210	32.76	132	81.51	132	79.20
SROSENBR	1000	20	1.18	19	1.77	14	6.85	11	5.92
TQUARTIC	1000	27	1.77	27	2.80	13	7.76	13	5.93
TRIDIA	1000	763	48.90	534	78.98	3	3.96	3	3.91

Table 1.2.Test results of L-BFGS-B, using the default option (primal method) for subspace minimization, and results of LANCELOT's SR1 and exact Hessian options, on unconstrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.
F1: Gradient stopping test (2) was not met but the final function value was greater than that obtained by LANCELOT SR1.
C2: The SR1 option of LANCELOT converged to a different solution point than the other methods.
F2: The SR1 option of LANCELOT could not satisfy the stopping test after 9999 function evaluations.

Varying m - Bound Constrained Problems

Problem	n	L-BFGS-B m=3 (Primal)		L-BFGS-B m=5 (Primal)		L-BFGS-B m=17 (Primal)		L-BFGS-B m=29 (Primal)	
		nfg	time	nfg	time	nfg	time	nfg	time
BDEXP	1000	15	1.91	15	2.31	16	3.50	16	3.61
BIGGS5	6	109	0.57	121	0.88	69	1.51	71	3.23
BQPGASIM	50	28	0.25	25	0.28	23	0.43	23	0.43
BQPGAUSS	2003	*F1	(3E–2)	*F1	(7E–3)	*C1	(4E–4)	**C1	(5E–5)
HATFLDC	25	25	0.14	23	0.19	23	0.41	23	0.36
HS110	50	2	0.01	2	0.02	2	0.02	2	0.02
HS45	5	11	0.02	11	0.03	11	0.01	11	0.01
JNLBRNGA	15625	389	763.79	332	740.33	296	1133.88	323	1758.70
JNLBRNGB	1024	569	65.13	424	62.73	426	125.17	447	228.05
LINVERSE	999	564	91.89	291	56.85	369	159.31	416	315.31
MAXLIKA	8	*F1	(5E–3)	1665	88.38	158	10.27	118	10.67
MCCORMCK	1000	15	2.00	15	1.85	15	2.05	15	2.04
NONSCOMP	1000	46	5.38	45	6.79	60	17.24	61	20.14
OBSTCLAE	5625	261	182.05	258	207.20	308	455.60	282	578.10
OBSTCLAL	1024	39	4.74	40	5.84	40	10.45	39	11.71
OBSTCLBL	1024	55	7.07	50	7.83	55	16.62	53	22.27
OBSTCLBM	15625	161	338.97	146	353.04	138	573.84	146	828.85
OBSTCLBU	1024	46	5.62	44	6.57	41	11.48	41	15.12
PALMER1A	6	*F1	(2E–1)	799	4.95	262	4.50	197	8.01
PALMER1E	8	*F1	(2E–1)	*F1	(7E–2)	290	5.06	254	10.81
PALMER2A	6	2888	16.26	518	3.67	182	4.12	170	9.69
PALMER2E	8	*F1	(1E–3)	*F1	(2E–3)	291	6.98	221	13.29
PALMER3A	6	2460	14.12	716	5.13	140	3.31	134	7.45
PALMER3E	8	*F1	(2E–3)	*F1	(4E–4)	221	3.59	182	7.50
PALMER4A	6	1985	11.38	483	3.30	128	2.82	90	4.32
PALMER4E	8	*F1	(5E–2)	*F1	(3E–3)	172	2.89	142	5.42
PROBPENL	500	3	0.11	3	0.10	3	0.11	3	0.10
S368	100	19	15.23	21	16.84	21	16.93	21	16.86
TORSION1	1024	60	7.38	43	6.35	32	8.16	33	9.51
TORSION2	1024	59	7.87	61	10.08	55	18.33	63	30.45
TORSION3	1024	27	2.86	23	2.76	22	3.61	22	3.65
TORSION4	1024	50	5.96	49	5.87	43	8.32	42	10.17
TORSION6	14884	309	565.25	362	707.22	360	1157.74	422	1994.78

Table 1.3. Test results of L-BFGS-B, using the default option (primal method) for subspace minimization and various values for m, on bound constrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.
F1: Gradient stopping test (2) was not met but the final function value was greater than that obtained by LANCELOT SR1.

Varying m -Unconstrained Problems

Problem	n	L-BFGS-B m=3 (Primal)		L-BFGS-B m=5 (Primal)		L-BFGS-B m=17 (Primal)		L-BFGS-B m=29 (Primal)	
		nfg	time	nfg	time	nfg	time	nfg	time
ARWHEAD	1000	12	0.95	13	1.09	**C1	(2E–5)	**C1	(2E–5)
BDQRTIC	100	124	1.34	101	1.28	47	1.29	39	1.59
BROYDN7D	1000	393	64.47	373	66.30	398	104.51	384	146.08
CRAGGLVY	1000	99	12.79	95	13.33	89	19.08	85	24.45
DIXMAANA	1500	11	1.09	12	1.34	13	1.66	13	1.61
DIXMAANB	1500	12	1.24	12	1.36	12	1.43	12	1.44
DIXMAANC	1500	14	1.47	14	1.61	14	1.85	14	1.81
DIXMAAND	1500	15	1.56	15	1.70	15	2.08	15	2.04
DIXMAANE	1500	214	23.53	188	24.28	169	41.07	166	60.31
DIXMAANF	1500	164	18.14	163	21.04	126	30.71	124	45.14
DIXMAANG	1500	191	20.93	158	20.38	127	30.94	132	47.22
DIXMAANH	1500	157	17.37	156	20.30	124	30.01	127	46.04
DIXMAANI	1500	828	97.87	1237	166.37	1066	273.08	922	364.27
DIXMAANK	1500	146	16.10	130	16.59	146	35.36	133	47.63
DIXMAANL	1500	164	17.93	134	16.93	120	28.04	125	44.08
DQDRTIC	1000	23	1.64	19	1.47	19	1.73	19	1.76
DQRTIC	500	43	1.28	43	1.46	43	2.96	43	4.06
EIGENALS	110	769	21.30	574	17.21	302	15.77	145	13.02
EIGENBLS	110	1445	39.91	1116	33.36	1041	55.73	870	86.35
EIGENCLS	462	2613	493.70	2900	563.81	2507	599.32	1969	593.89
ENGVAL1	1000	23	1.78	23	2.02	20	2.38	20	2.39
FLETCHBV	100	**C1	(4E–3)	**C1	(2E–0)	**C1	(1E–0)	**C1	(4E–1)
FREUROTH	1000	**C1	(4E–5)	**C1	(2E–5)	**C1	(1E–3)	38	6.40
GENROSE	500	1323	57.99	1244	60.86	1315	116.82	1306	198.67
MOREBV	1000	73	5.50	79	6.85	77	12.22	76	18.04
NONDIA	1000	21	1.48	23	1.79	23	2.56	23	2.67
NONDQUAR	100	866	6.96	1001	10.09	828	25.82	588	43.50
PENALTY1	1000	60	3.26	60	3.91	60	7.58	60	10.96
PENALTY3	100	**C1	(9E–3)	**C1	(3E–3)	**C1	(3E–3)	**C1	(6E–4)
QUARTC	1000	47	2.68	47	3.10	47	5.86	47	7.62
SINQUAD	1000	211	17.45	183	17.17	210	32.76	231	52.93
SROSENBR	1000	18	0.90	20	1.18	19	1.77	19	1.81
TQUARTIC	1000	23	1.34	27	1.77	27	2.80	27	2.97
TRIDIA	1000	882	44.16	763	48.90	534	78.98	474	120.90

Table 1.4. Test results of L-BFGS-B, using the default option (primal method) for subspace minimization and various values of m, on unconstrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.

Comparing Dual and CG Options on Bound Constrained Problems

Problem	n	L-BFGS-B m=5 (Dual)		L-BFGS-B m=17 (Dual)		L-BFGS-B m=5 (CG)		L-BFGS-B m=17 (CG)	
		nfg	time	nfg	time	nfg	time	nfg	time
BDEXP	1000	15	1.44	16	1.91	15	2.38	16	3.48
BIGGS5	6	118	0.83	69	1.57	110	0.80	61	0.97
BQPGASIM	50	25	0.23	23	0.40	26	0.32	23	0.51
BQPGAUSS	2003	*F1	(3E–3)	*C1	(1E–4)	*F1	(7E–2)	*C1	(3E–4)
HATFLDC	25	23	0.15	23	0.26	24	0.23	24	0.45
HS110	50	2	0.02	2	0.03	2	0.02	2	0.01
HS45	5	11	0.01	11	0.02	11	0.02	11	0.03
JNLBRNGA	15625	325	770.36	334	1396.17	557	1419.78	400	2192.56
JNLBRNGB	1024	440	63.60	432	121.42	532	89.31	410	219.70
LINVERSE	999	303	52.11	373	126.39	539	104.97	124	37.68
MAXLIKA	8	2199	121.00	155	10.13	601	33.20	133	8.19
MCCORMCK	1000	15	1.55	15	1.66	16	2.35	16	2.76
NONSCOMP	1000	45	3.81	60	8.77	41	4.27	36	6.83
OBSTCLAE	5625	258	210.71	299	439.11	315	258.82	295	438.76
OBSTCLAL	1024	40	5.57	40	9.60	39	5.75	42	12.90
OBSTCLBL	1024	50	8.44	55	18.23	53	9.00	54	19.16
OBSTCLBM	15625	150	349.89	136	539.99	264	646.57	256	1046.88
OBSTCLBU	1024	44	7.08	41	9.92	49	7.18	48	12.99
PALMER1A	6	1342	7.44	263	4.04	7367	44.73	390	5.17
PALMER1E	8	*F1	(2E–3)	309	5.23	*F1	(1E–2)	412	6.91
PALMER2A	6	484	3.08	174	3.51	2197	12.72	159	2.07
PALMER2E	8	*F1	(2E–3)	289	6.00	*F1	(1E–2)	505	8.48
PALMER3A	6	600	3.83	143	2.93	5046	38.02	220	3.97
PALMER3E	8	*F1	(2E–3)	224	3.45	*F1	(7E–2)	313	4.14
PALMER4A	6	584	3.72	128	2.58	909	5.43	155	1.95
PALMER4E	8	*F1	(2E–3)	198	3.16	*F1	(6E–3)	279	4.57
PROBPENL	500	3	0.13	3	0.13	3	0.11	3	0.12
S368	100	21	16.83	21	16.89	27	21.66	28	22.50
TORSION1	1024	43	6.72	32	8.45	63	9.47	44	13.20
TORSION2	1024	61	9.83	55	16.74	73	11.69	58	16.04
TORSION3	1024	23	3.98	22	6.12	24	3.02	22	4.28
TORSION4	1024	49	7.32	43	12.03	50	5.96	42	8.13
TORSION6	14884	331	812.97	394	1848.57	357	651.52	354	1005.46

Table 1.5. Test results of L-BFGS-B method, using dual and CG methods for subspace minimization, on bound constrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.
F1: Gradient stopping test (2) was not met but the final function value was greater than that obtained by LANCELOT SR1.

Comparing Dual and CG Options on Unconstrained Problems

Problem	n	L-BFGS-B m=5 (Dual)		L-BFGS-B m=17 (Dual)		L-BFGS-B m=5 (CG)		L-BFGS-B m=17 (CG)	
		nfg	time	nfg	time	nfg	time	nfg	time
ARWHEAD	1000	13	1.06	**C1	(2E–5)	13	1.10	12	1.13
BDQRTIC	100	101	1.24	47	1.27	90	1.59	44	1.80
BROYDN7D	1000	373	64.39	388	95.45	378	92.53	388	189.17
CRAGGLVY	1000	95	13.27	89	17.84	86	17.47	79	35.40
DIXMAANA	1500	12	1.27	13	1.53	12	1.34	13	1.61
DIXMAANB	1500	12	1.27	12	1.38	12	1.28	12	1.34
DIXMAANC	1500	14	1.58	14	1.77	14	1.58	14	1.75
DIXMAAND	1500	15	1.68	15	1.97	16	1.83	16	2.09
DIXMAANE	1500	187	23.10	170	38.32	213	54.04	172	166.98
DIXMAANF	1500	163	19.97	126	27.75	165	41.75	129	124.67
DIXMAANG	1500	153	19.06	127	28.69	147	36.02	127	114.97
DIXMAANH	1500	156	19.42	124	27.79	152	38.43	131	120.53
DIXMAANI	1500	1017	131.59	994	238.46	1075	298.41	1206	1679.64
DIXMAANK	1500	130	16.01	146	32.64	163	41.04	127	121.68
DIXMAANL	1500	134	16.29	120	25.91	213	56.73	122	120.51
DQDRTIC	1000	19	1.45	19	1.63	23	2.30	18	1.95
DQRTIC	500	43	1.43	43	2.80	44	1.58	44	3.02
EIGENALS	110	548	16.08	373	19.05	575	20.08	294	24.23
EIGENBLS	110	1156	33.00	1113	50.62	2221	79.27	1071	102.08
EIGENCLS	462	2777	536.89	2572	600.46	3329	736.95	2469	1239.65
ENGVAL1	1000	23	1.96	20	2.24	23	2.38	22	3.31
FLETCHBV	100	**C1	(2E–2)	**C1	(2E–0)	**C1	(2E–1)	**C1	(3E–0)
FREUROTH	1000	**C1	(2E–5)	**C1	(1E–3)	71	7.03	**C1	(6E–5)
GENROSE	500	1269	59.93	1321	115.45	1426	106.84	1400	315.76
MOREBV	1000	79	6.55	77	11.46	74	13.67	86	62.30
NONDIA	1000	23	1.77	23	2.42	18	1.12	18	1.13
NONDQUAR	100	1162	11.15	811	24.85	992	16.10	834	86.53
PENALTY1	1000	60	3.72	60	6.97	72	4.42	67	6.56
PENALTY3	100	**C1	(4E–3)	**C1	(1E–3)	**C1	(5E–3)	**C1	(2E–3)
QUARTC	1000	47	3.01	47	5.35	47	3.35	48	6.16
SINQUAD	1000	196	17.62	209	30.62	148	14.37	174	26.46
SROSENBR	1000	20	1.14	19	1.64	19	1.22	19	1.60
TQUARTIC	1000	27	1.73	27	2.62	30	2.02	30	2.83
TRIDIA	1000	767	45.36	594	80.30	1299	182.94	610	399.62

Table 1.6. Test results of L-BFGS-B method, using dual and CG methods for subspace minimization, on unconstrained problems from the CUTE collection.

* : Termination because the number of function evaluations reached 9999.
** : Termination because the code could make no further progress in reducing f.
(In cases ** and * the value in parentheses is the norm of the projected gradient at the final iterate.)
C1: Gradient stopping test (2) was not met but the final function value was at least as good as that obtained by LANCELOT SR1.

Acknowledgements. The authors would like to thank Brett Averick and Jorge Moré for their help and suggestions. This code follows many of the ideas and the style of their MINPACK-2 codes [2].

* References

[1] B. M. Averick and J. J. Moré, (1992). Private communication.

[2] B. M. Averick and J. J. Moré, "The MINPACK-2 package", in preparation.

[3] D.P. Bertsekas, "Projected Newton methods for optimization problems with simple constraints", *SIAM J. Control and Optimization* 20 (1982), pp. 221-246.

[4] I. Bongartz, A.R. Conn, N.I.M. Gould, Ph.L. Toint (1993). "CUTE: constrained and unconstrained testing environment", Research Report, IBM T.J. Watson Research Center, Yorktown, USA.

[5] Buckley, A. and LeNir, A, "BBVSCG –A variable storage algorithm for function minimization", *ACM Transactions on Mathematical Software* 11/2 (1985), pp. 103-119.

[6] Buckley, A. "Remark on algorithm 630", *ACM Transactions on Mathematical Software* 15,3 (1989), pp. 262-274.

[7] R. H. Byrd, J. Nocedal and R. B. Schnabel, "Representation of quasi-Newton matrices and their use in limited memory methods", *Mathematical Programming* 63, 4, 1994, pp. 129-156.

[8] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu. "A limited memory algorithm for bound constrained optimization" Tech. Report, EECS Department, Northwestern University, 1993, to appear in *SIAM Journal on Scientific Computing.*

[9] A. R. Conn, N. I. M. Gould, and PH. L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables", *Mathematics of Computation.* Vol. 50, No 182 (1988), pp. 399-430.

[10] A.R. Conn, N.I.M. Gould, Ph.L. Toint (1992). "LANCELOT: a FORTRAN package for large-scale nonlinear optimization (Release A)", Number 17 in Springer Series in Computational Mathematics, Springer-Verlag, New York.

[11] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, N.J., Prentice-Hall, 1983.

[12] *Harwell Subroutine Library, Release 10* (1990). Advanced Computing Department, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, United Kingdom.

[13] J.C. Gilbert and C. Lemaréchal, "Some numerical experiments with variable storage quasi-Newton algorithms," *Mathematical Programming* 45 (1989), pp. 407–436.

[14] P. E. Gill, W Murray and M. H. Wright, *Practical Optimization*, London, Academic Press, 1981.

[15] E. S. Levitin and B. T. Polyak, "Constrained minimization problems", *USSR Comput. Math. and Math. Phys.* 6 (1966), pp. 1-50.

[16] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization methods", *Mathematical Programming* 45 (1989), pp. 503-528.

[17] J. J. Moré and D.J. Thuente (1990), "On line search algorithms with guaranteed sufficient decrease", Mathematics and Computer Science Division Preprint MCS-P153-0590, Argonne National Laboratory (Argonne, IL).

[18] J. J. Moré and G. Toraldo, "Algorithms for bound constrained quadratic programming problems", *Numer. Math.* 55 (1989), pp. 377-400.

[19] J. Nocedal, "Updating quasi-Newton matrices with limited storage", *Mathematics of Computation* 35 (1980), pp. 773-782.