L-BFGS-B

3.0

Generated by Doxygen 1.9.1

1 L-BFGS-B

Software for Large-scale Bound-constrained Optimization

L-BFGS-B is a limited-memory quasi-Newton code for bound-constrained optimization, i.e., for problems where the only constraints are of the form `l <= x <= u`. 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.

Authors

- Ciyou Zhu
- Richard Byrd
- Jorge Nocedal
- Jose Luis Morales

Related Publications

- R. H. Byrd, P. Lu, J. Nocedal and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization (1995), SIAM Journal on Scientific and Statistical Computing, Vol. 16, Num. 5, pp. 1190-1208
- C. Zhu, R. H. Byrd and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization (1997), ACM Transactions on Mathematical Software, Vol. 23, Num. 4, pp. 550-560
- J.L. Morales and J. Nocedal. L-BFGS-B: Remark on Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization (2011), ACM Transactions on Mathematical Software, Vol. 38, Num. 1

- R. H. Byrd, J. Nocedal and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods (1994), Mathematical Programming, Vol. 63, pp. 129-156

Note that the subspace minimization in the LBFGSpp implementation is an exact minimization subject to the bounds, based on the BOXCQP algorithm:

- C. Voglis and I. E. Lagaris, BOXCQP: An Algorithm for Bound Constrained Convex Quadratic Problems (2004), 1st International Conference "From Scientific Computing to Computational Engineering", Athens, Greece

For an eagle-eye overview of L-BFGS-B and the genealogy BFGS->L-BFGS->L-BFGS-B, see Henao's Master's thesis.

Related Software

- wilmerhenao/L-BFGS-B-NS: An L-BFGS-B-NS Optimizer for Non-Smooth Functions
- pcarbo/lbfgsb-matlab: A MATLAB interface for L-BFGS-B
- bgranzow/L-BFGS-B: A pure Matlab implementation of L-BFGS-B (LBFGSB)
- constantino-garcia/lbfgsb_cpp_wrapper: A simple C++ wrapper around the original Fortran L-BGSG-B routine
- yixuan/LBFGSpp: A header-only C++ library for L-BFGS and L-BFGS-B algorithms
- chokkan/liblbfgs: libLBFGS: a library of Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)
- mkobos/lbfgsb_wrapper: Java wrapper for the Fortran L-BFGS-B algorithm
- yuhonglin/Lbfgsb.jl: A Julia wrapper of the l-bfgs-b fortran library
- Gnimuc/LBFGSB.jl: Julia wrapper for L-BFGS-B Nonlinear Optimization Code
- afbarnard/go-lbfgsb: L-BFGS-B optimization for Go, C, Fortran 2003
- nepluno/lbfgsb-gpu: An open source library for the GPU-implementation of L-BFGS-B algorithm
- Chris00/L-BFGS-ocaml: OCaml bindings for L-BFGS
- dwicke/L-BFGS-B-Lua: L-BFGS-B lua wrapper around a L-BFGS-B C implementation
- avieira/python_lbfgsb: Pure Python-based L-BFGS-B implementation
- ybyygu/rust-lbfgsb: Ergonomic bindings to L-BFGS-B code for Rust
- rforge/lbfgsb3c: Limited Memory BFGS Minimizer with Bounds on Parameters with optim() 'C' Interface for R
- florafauna/optimParallel-python: A parallel version of 'scipy.optimize.minimize(method='L-BFGS-B')`

Notes on this repository

I (J. Schilling) took the freedom to

- put the L-BFGS-B code obtained from the original website up in this repository,
- divide the subroutines and functions into separate files,
- convert parts of the documentation into a format understandable to doxygen and
- replace the included BLAS/LINPACK routines with calls to user-provided BLAS/LAPACK routines. To be precise, the calls to LINPACK's dtrsl were replace with calls to LAPACK's dtrsm and the calls to LINPACK's dpofa were replaces with calls to LAPACK's dpotrf.

Building

A CMake setup is provided for L-BFGS-B in this repository. External modules for BLAS and LAPACK have to be installed on your system. Then, building the shared library liblbfgsb.so and the examples driver*.f and driver*.f90 works as follows:

```
> mkdir build
> cd build
> cmake ..
> make -j
```

The resulting shared library and the driver executables can be found in the build directory.

Concluding remarks

The current release is version 3.0. The distribution file was last changed on 02/08/11.

This work was in no way intending to infringe any copyrights or take credit for others' work. Feel free to contact me at any time in case you noticed something against the rules. Above documentation is obtained from the archived version of the original manual.

A PDF version of the documentation is available here: L-BFGS-B.pdf.

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 File Documentation

3.1 src/active.f File Reference

Functions/Subroutines

- subroutine active (n, l, u, nbd, x, iwhere, iprint, prjctd, cnstnd, boxed)

  This subroutine initializes iwhere and projects the initial x to the feasible set if necessary.

3.1.1 Function/Subroutine Documentation

3.1.1.1 active()

```
subroutine active (
        integer n,
        double precision, dimension(n) l,
        double precision, dimension(n) u,
        integer, dimension(n) nbd,
        double precision, dimension(n) x,
        integer, dimension(n) iwhere,
        integer iprint,
        logical prjctd,
        logical cnstnd,
        logical boxed )
```

This subroutine initializes iwhere and projects the initial x to the feasible set if necessary.

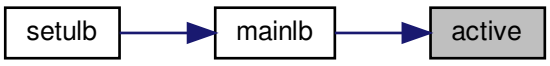
Parameters

n	number of parameters
l	lower bounds on parameters
u	upper bounds on parameters
nbd	indicates which bounds are present
x	position
iwhere	On entry iwhere is unspecified. On exit: iwhere(i)= • -1 if x(i) has no bounds • 3 if l(i)=u(i), • 0 otherwise. In cauchy, iwhere is given finer gradations.
iprint	console output flag
prjctd	TODO
cnstnd	TODO
boxed	TODO

Definition at line 32 of file active.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.2 src/bmv.f File Reference

Functions/Subroutines

- subroutine bmv (m, sy, wt, col, v, p, info)

  This subroutine computes the product of the 2m x 2m middle matrix in the compact L-BFGS formula of B and a 2m vector v.

3.2.1 Function/Subroutine Documentation

3.2.1.1 bmv()

```
subroutine bmv (
        integer m,
        double precision, dimension(m, m) sy,
        double precision, dimension(m, m) wt,
        integer col,
        double precision, dimension(2*col) v,
        double precision, dimension(2*col) p,
        integer info )
```

This subroutine computes the product of the 2m x 2m middle matrix in the compact L-BFGS formula of B and a 2m vector v; it returns the product in p.

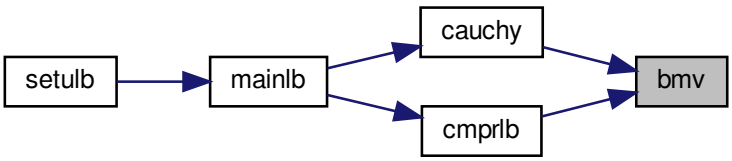
Parameters

m	On entry m is the maximum number of variable metric corrections used to define the limited memory matrix. On exit m is unchanged.
sy	On entry sy specifies the matrix S'Y. On exit sy is unchanged.
wt	On entry wt specifies the upper triangular matrix J' which is the Cholesky factor of (thetaS'S+LD^(-1)L'). On exit wt is unchanged.
col	On entry col specifies the number of s-vectors (or y-vectors) stored in the compact L-BFGS formula. On exit col is unchanged.
v	On entry v specifies vector v. On exit v is unchanged.
p	On entry p is unspecified. On exit p is the product Mv.
info	On entry info is unspecified. On exit info = • 0 for normal return, • nonzero for abnormal return when the system to be solved by dtrsl is singular.

Definition at line 35 of file bmv.f.

Referenced by cauchy(), and cmprlb().

Here is the caller graph for this function:



3.3 src/cauchy.f File Reference

Functions/Subroutines

- subroutine cauchy (n, x, l, u, nbd, g, iorder, iwhere, t, d, xcp, m, wy, ws, sy, wt, theta, col, head, p, c, wbp, v, nseg, iprint, sbgnrm, info, epsmch)

  Compute the Generalized Cauchy Point along the projected gradient direction.

3.3.1 Function/Subroutine Documentation

3.3.1.1 cauchy() `subroutine cauchy (`

```
integer n,
double precision, dimension(n) x,
double precision, dimension(n) l,
double precision, dimension(n) u,
integer, dimension(n) nbd,
double precision, dimension(n) g,
integer, dimension(n) iorder,
integer, dimension(n) iwhere,
double precision, dimension(n) t,
double precision, dimension(n) d,
double precision, dimension(n) xcp,
integer m,
double precision, dimension(n, col) wy,
double precision, dimension(n, col) ws,
double precision, dimension(m, m) sy,
double precision, dimension(m, m) wt,
double precision theta,
integer col,
integer head,
double precision, dimension(2*m) p,
double precision, dimension(2*m) c,
double precision, dimension(2*m) wbp,
double precision, dimension(2*m) v,
integer nseg,
integer iprint,
double precision sbgnrm,
integer info,
double precision epsmch )
```

For given x, l, u, g (with sbgnrm $>$ 0), and a limited memory BFGS matrix B defined in terms of matrices WY, WS, WT, and scalars head, col, and theta, this subroutine computes the generalized Cauchy point (GCP), defined as the first local minimizer of the quadratic

```
Q(x + s) = g's + 1/2 s'Bs
```

along the projected gradient direction P(x-tg,l,u). The routine returns the GCP in xcp.

Parameters

n	On entry n is the dimension of the problem. On exit n is unchanged.

Parameters

x	On entry x is the starting point for the GCP computation. On exit x is unchanged.
l	On entry l is the lower bound of x. On exit l is unchanged.
u	On entry u is the upper bound of x. On exit u is unchanged.
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: 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, and • 3 if x(i) has only an upper bound. On exit nbd is unchanged.
g	On entry g is the gradient of f(x). g must be a nonzero vector. On exit g is unchanged.
iorder	iorder will be used to store the breakpoints in the piecewise linear path and free variables encountered. On exit, • iorder(1),...,iorder(nleft) are indices of breakpoints which have not been encountered; • iorder(nleft+1),...,iorder(nbreak) are indices of encountered breakpoints; and • iorder(nfree),...,iorder(n) are indices of variables which have no bound constraits along the search direction.
iwhere	On entry iwhere indicates only the permanently fixed (iwhere=3) or free (iwhere= -1) components of x. On exit iwhere records the status of the current x variables. iwhere(i)= • -3 if x(i) is free and has bounds, but is not moved • 0 if x(i) is free and has bounds, and is moved • 1 if x(i) is fixed at l(i), and l(i) .ne. u(i) • 2 if x(i) is fixed at u(i), and u(i) .ne. l(i) • 3 if x(i) is always fixed, i.e., u(i)=x(i)=l(i) • -1 if x(i) is always free, i.e., it has no bounds.
t	working array; will be used to store the break points.
d	the Cauchy direction P(x-tg)-x
xcp	returns the GCP on exit
m	On entry m is the maximum number of variable metric corrections used to define the limited memory matrix. On exit m is unchanged.
ws	On entry this stores S, a set of s-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
wy	On entry this stores Y, a set of y-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
sy	On entry this stores S'Y, that defines the limited memory BFGS matrix. On exit this array is unchanged.

Parameters

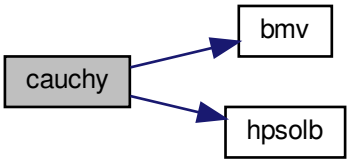
wt	On entry this stores the Cholesky factorization of (theta∗S'S+LD^(-1)L'), that defines the limited memory BFGS matrix. On exit this array is unchanged.
theta	On entry theta is the scaling factor specifying B_0 = theta I. On exit theta is unchanged.
col	On entry col is the actual number of variable metric corrections stored so far. On exit col is unchanged.
head	On entry head is the location of the first s-vector (or y-vector) in S (or Y). On exit col is unchanged.
p	will be used to store the vector p = W^(T)d.
c	will be used to store the vector c = W^(T)(xcp-x).
wbp	will be used to store the row of W corresponding to a breakpoint.
v	working array
nseg	On exit nseg records the number of quadratic segments explored in searching for the GCP.
iprint	variable that must be set by the user. It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.
sbgnrm	On entry sbgnrm is the norm of the projected gradient at x. On exit sbgnrm is unchanged.
info	On entry info is 0. On exit info = • 0 for normal return, • = nonzero for abnormal return when the the system used in routine bmv is singular.
epsmch	machine precision epsilon

Definition at line 130 of file cauchy.f.

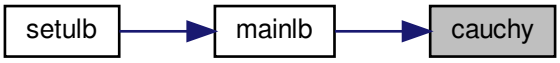
References bmv(), and hpsolb().

Referenced by mainlb().

Here is the call graph for this function:



Here is the caller graph for this function:



3.4 src/cmprlb.f File Reference

Functions/Subroutines

- subroutine cmprlb (n, m, x, g, ws, wy, sy, wt, z, r, wa, index, theta, col, head, nfree, cnstnd, info)

  This subroutine computes r=-Z'B(xcp-xk)-Z'g by using wa(2m+1)=W'(xcp-x) from subroutine cauchy.

3.4.1 Function/Subroutine Documentation

3.4.1.1 cmprlb()

```
subroutine cmprlb (
        integer n,
        integer m,
        double precision, dimension(n) x,
        double precision, dimension(n) g,
        double precision, dimension(n, m) ws,
        double precision, dimension(n, m) wy,
        double precision, dimension(m, m) sy,
        double precision, dimension(m, m) wt,
        double precision, dimension(n) z,
        double precision, dimension(n) r,
        double precision, dimension(4*m) wa,
        integer, dimension(n) index,
```

```
double precision theta,
integer col,
integer head,
integer nfree,
logical cnstnd,
integer info )
```

This subroutine computes r=-Z'B(xcp-xk)-Z'g by using wa(2m+1)=W'(xcp-x) from subroutine cauchy.

Parameters

n	number of parameters
m	history size of Hessian approximation
x	position
g	gradient
ws	part of L-BFGS matrix
wy	part of L-BFGS matrix
sy	part of L-BFGS matrix
wt	part of L-BFGS matrix
z	TODO
r	TODO
wa	TODO
index	TODO
theta	TODO
col	TODO
head	TODO
nfree	TODO
cnstnd	TODO
info	TODO

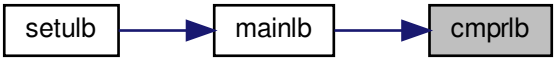
Definition at line 27 of file cmprlb.f.

References bmv().

Referenced by mainlb().

Here is the call graph for this function:

Here is the caller graph for this function:



3.5 src/dcsrch.f File Reference

Functions/Subroutines

- subroutine dcsrch (f, g, stp, ftol, gtol, xtol, stpmin, stpmax, task, isave, dsave)

  This subroutine finds a step that satisfies a sufficient decrease condition and a curvature condition.

3.5.1 Function/Subroutine Documentation

3.5.1.1 dcsrch() subroutine dcsrch (

```
double precision f,
double precision g,
double precision stp,
double precision ftol,
double precision gtol,
double precision xtol,
double precision stpmin,
double precision stpmax,
character*(*) task,
integer, dimension(2) isave,
double precision, dimension(13) dsave )
```

This subroutine finds a step that satisfies a sufficient decrease condition and a curvature condition.

Each call of the subroutine updates an interval with endpoints stx and sty. The interval is initially chosen so that it contains a minimizer of the modified function

```
psi(stp) = f(stp) - f(0) - ftol*stp*f'(0).
```

If psi(stp) <= 0 and f'(stp) >= 0 for some step, then the interval is chosen so that it contains a minimizer of f.

The algorithm is designed to find a step that satisfies the sufficient decrease condition

```
f(stp) <= f(0) + ftol*stp*f'(0),
```

and the curvature condition

```
abs(f'(stp)) <= gtol*abs(f'(0)).
```

If ftol is less than gtol and if, for example, the function is bounded below, then there is always a step which satisfies both conditions.

If no step can be found that satisfies both conditions, then the algorithm stops with a warning. In this case stp only satisfies the sufficient decrease condition.

A typical invocation of dcsrch has the following outline:

```
   task = 'START'
10 continue
   call dcsrch( ... )
   if (task .eq. 'FG') then
     evaluate the function and the gradient at stp
   goto 10
   end if
```

NOTE: The user must no alter work arrays between calls.

Parameters

f	On initial entry f is the value of the function at 0. On subsequent entries f is the value of the function at stp. On exit f is the value of the function at stp.
g	On initial entry g is the derivative of the function at 0. On subsequent entries g is the derivative of the function at stp. On exit g is the derivative of the function at stp.
stp	On entry stp is the current estimate of a satisfactory step. On initial entry, a positive initial estimate must be provided. On exit stp is the current estimate of a satisfactory step if task = 'FG'. If task = 'CONV' then stp satisfies the sufficient decrease and curvature condition.
ftol	On entry ftol specifies a nonnegative tolerance for the sufficient decrease condition. On exit ftol is unchanged.
gtol	On entry gtol specifies a nonnegative tolerance for the curvature condition. On exit gtol is unchanged.
xtol	On entry xtol specifies a nonnegative relative tolerance for an acceptable step. The subroutine exits with a warning if the relative difference between sty and stx is less than xtol. On exit xtol is unchanged.
stpmin	On entry stpmin is a nonnegative lower bound for the step. On exit stpmin is unchanged.
stpmax	On entry stpmax is a nonnegative upper bound for the step. On exit stpmax is unchanged.
task	On initial entry task must be set to 'START'. On exit task indicates the required action: • If task(1:2) = 'FG' then evaluate the function and derivative at stp and call dcsrch again. • If task(1:4) = 'CONV' then the search is successful. • If task(1:4) = 'WARN' then the subroutine is not able to satisfy the convergence conditions. The exit value of stp contains the best point found during the search. • If task(1:5) = 'ERROR' then there is an error in the input arguments. On exit with convergence, a warning or an error, the variable task contains additional information.
isave	work array
dsave	work array

Definition at line 94 of file dcsrch.f.

References dcstep().

Referenced by lnsrlb().

Here is the call graph for this function:

Here is the caller graph for this function:

3.6 src/dcstep.f File Reference

Functions/Subroutines

- subroutine dcstep (stx, fx, dx, sty, fy, dy, stp, fp, dp, brackt, stpmin, stpmax)

  This subroutine computes a safeguarded step for a search procedure and updates an interval that contains a step that satisfies a sufficient decrease and a curvature condition.

3.6.1 Function/Subroutine Documentation

3.6.1.1 dcstep()

```
subroutine dcstep (
        double precision stx,
        double precision fx,
        double precision dx,
        double precision sty,
        double precision fy,
        double precision dy,
        double precision stp,
        double precision fp,
        double precision dp,
        logical brackt,
        double precision stpmin,
        double precision stpmax )
```

This subroutine computes a safeguarded step for a search procedure and updates an interval that contains a step that satisfies a sufficient decrease and a curvature condition.

The parameter stx contains the step with the least function value. If brackt is set to .true. then a minimizer has been bracketed in an interval with endpoints stx and sty. The parameter stp contains the current step. The subroutine assumes that if brackt is set to .true. then

```
min(stx,sty) < stp < max(stx,sty),
```

and that the derivative at stx is negative in the direction of the step.

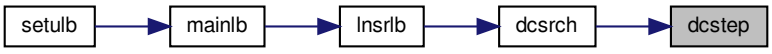
Parameters

stx	On entry stx is the best step obtained so far and is an endpoint of the interval that contains the minimizer. On exit stx is the updated best step.
fx	On entry fx is the function at stx. On exit fx is the function at stx.
dx	On entry dx is the derivative of the function at stx. The derivative must be negative in the direction of the step, that is, dx and stp - stx must have opposite signs. On exit dx is the derivative of the function at stx.
sty	On entry sty is the second endpoint of the interval that contains the minimizer. On exit sty is the updated endpoint of the interval that contains the minimizer.
fy	On entry fy is the function at sty. On exit fy is the function at sty.
dy	On entry dy is the derivative of the function at sty. On exit dy is the derivative of the function at the exit sty.
stp	On entry stp is the current step. If brackt is set to .true. then on input stp must be between stx and sty. On exit stp is a new trial step.
fp	On entry fp is the function at stp. On exit fp is unchanged.
dp	On entry dp is the the derivative of the function at stp. On exit dp is unchanged.
brackt	On entry brackt specifies if a minimizer has been bracketed. Initially brackt must be set to .false. On exit brackt specifies if a minimizer has been bracketed. When a minimizer is bracketed brackt is set to .true.
stpmin	On entry stpmin is a lower bound for the step. On exit stpmin is unchanged.
stpmax	On entry stpmax is an upper bound for the step. On exit stpmax is unchanged.

Definition at line 66 of file dcstep.f.

Referenced by dcsrch().

Here is the caller graph for this function:



3.7 src/errclb.f File Reference

Functions/Subroutines

- subroutine errclb (n, m, factr, l, u, nbd, task, info, k)

  This subroutine checks the validity of the input data.

3.7.1 Function/Subroutine Documentation

3.7.1.1 errclb() `subroutine errclb (`

```
integer n,
integer m,
double precision factr,
double precision, dimension(n) l,
double precision, dimension(n) u,
integer, dimension(n) nbd,
character*60 task,
integer info,
integer k )
```

This subroutine checks the validity of the input data.

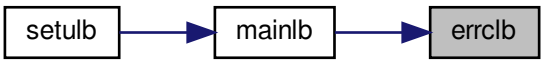
Parameters

n	number of parameters
m	history size of approximated Hessian
factr	convergence criterion on function value
l	lower bounds for parameters
u	upper bounds for parameters
nbd	indicates which bounds are present
task	if an error occurs, contains a human-readable error message
info	=0 on success; =-6 if nbd(k) was invalid; =-7 if both limits are given but l(k) $>$ u(k)
k	index of last errournous parameter

Definition at line 16 of file errclb.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.8 src/formk.f File Reference

Functions/Subroutines

- subroutine formk (n, nsub, ind, nenter, ileave, indx2, iupdat, updatd, wn, wn1, m, ws, wy, sy, theta, col, head, info)

  Forms the LEL^T factorization of the indefinite matrix K.

3.8.1 Function/Subroutine Documentation

3.8.1.1 formk()

```
subroutine formk (
        integer n,
        integer nsub,
        integer, dimension(n) ind,
        integer nenter,
        integer ileave,
        integer, dimension(n) indx2,
        integer iupdat,
        logical updatd,
        double precision, dimension(2*m, 2*m) wn,
        double precision, dimension(2*m, 2*m) wn1,
        integer m,
        double precision, dimension(n, m) ws,
        double precision, dimension(n, m) wy,
        double precision, dimension(m, m) sy,
        double precision theta,
        integer col,
        integer head,
        integer info )
```

This subroutine forms the LEL^T factorization of the indefinite matrix

K = [-D -Y'ZZ'Y/theta L_a'-R_z'] [L_a -R_z theta*S'AA'S] where E = [-I 0] [0 I]

The matrix K can be shown to be equal to the matrix M^[-1]N occurring in section 5.1 of [1], as well as to the matrix Mbar^[-1] Nbar in section 5.3.

Parameters

n	On entry n is the dimension of the problem. On exit n is unchanged.
nsub	On entry nsub is the number of subspace variables in free set. On exit nsub is not changed.
ind	On entry ind specifies the indices of subspace variables. On exit ind is unchanged.
nenter	On entry nenter is the number of variables entering the free set. On exit nenter is unchanged.
ileave	On entry indx2(ileave),...,indx2(n) are the variables leaving the free set. On exit ileave is unchanged.
indx2	On entry indx2(1),...,indx2(nenter) are the variables entering the free set, while indx2(ileave),...,indx2(n) are the variables leaving the free set. On exit indx2 is unchanged.
iupdat	On entry iupdat is the total number of BFGS updates made so far. On exit iupdat is unchanged.
updatd	On entry 'updatd' is true if the L-BFGS matrix is updated. On exit 'updatd' is unchanged.
wn	On entry wn is unspecified. On exit the upper triangle of wn stores the LEL^T factorization of the 2*col x 2*col indefinite matrix [-D -Y'ZZ'Y/theta L_a'-R_z'] [L_a -R_z theta*S'AA'S]

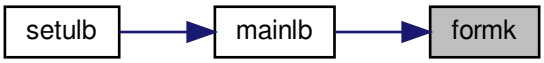
Parameters

wn1	On entry wn1 stores the lower triangular part of [Y' ZZ'Y L_a'+R_z'] [L_a+R_z S'AA'S] in the previous iteration. On exit wn1 stores the corresponding updated matrices. The purpose of wn1 is just to store these inner products so they can be easily updated and inserted into wn.
m	On entry m is the maximum number of variable metric corrections used to define the limited memory matrix. On exit m is unchanged.
ws	On entry this stores S, a set of s-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
wy	On entry this stores Y, a set of y-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
sy	On entry this stores S'Y, that defines the limited memory BFGS matrix. On exit this array is unchanged.
theta	On entry theta is the scaling factor specifying B_0 = theta I. On exit theta is unchanged.
col	On entry col is the actual number of variable metric corrections stored so far. On exit col is unchanged.
head	On entry head is the location of the first s-vector (or y-vector) in S (or Y). On exit col is unchanged.
info	On entry info is unspecified. On exit info • = 0 for normal return; • = -1 when the 1st Cholesky factorization failed; • = -2 when the 2st Cholesky factorization failed.

Definition at line 89 of file formk.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.9 src/formt.f File Reference

Functions/Subroutines

- subroutine formt (m, wt, sy, ss, col, theta, info)

  Forms the upper half of the pos. def. and symm. T.

3.9.1 Function/Subroutine Documentation

3.9.1.1 formt()

```
subroutine formt (
        integer m,
        double precision, dimension(m, m) wt,
        double precision, dimension(m, m) sy,
        double precision, dimension(m, m) ss,
        integer col,
        double precision theta,
        integer info )
```

This subroutine forms the upper half of the pos. def. and symm. T = theta∗SS + L∗D^(-1)∗L', stores T in the upper triangle of the array wt, and performs the Cholesky factorization of T to produce J∗J', with J' stored in the upper triangle of wt.

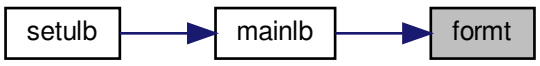
Parameters

m	history size of approximated Hessian
wt	part of L-BFGS matrix
sy	part of L-BFGS matrix
ss	part of L-BFGS matrix
col	On entry col is the actual number of variable metric corrections stored so far. On exit col is unchanged.
theta	On entry theta is the scaling factor specifying B_0 = theta I. On exit theta is unchanged.
info	error/success indicator

Definition at line 21 of file formt.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.10 src/freev.f File Reference

Functions/Subroutines

- subroutine freev (n, nfree, index, nenter, ileave, indx2, iwhere, wrk, updatd, cnstnd, iprint, iter)

  This subroutine counts the entering and leaving variables when iter > 0, and finds the index set of free and active variables at the GCP.

3.10.1 Function/Subroutine Documentation

3.10.1.1 freev()

```
subroutine freev (
        integer n,
        integer nfree,
        integer, dimension(n) index,
        integer nenter,
        integer ileave,
        integer, dimension(n) indx2,
        integer, dimension(n) iwhere,
        logical wrk,
        logical updatd,
        logical cnstnd,
        integer iprint,
        integer iter )
```

This subroutine counts the entering and leaving variables when iter $>$ 0, and finds the index set of free and active variables at the GCP.

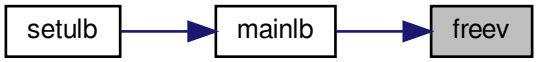
Parameters

n	number of parameters
nfree	number of free parameters, i.e., those not at their bounds
index	for i=1,...,nfree, index(i) are the indices of free variables for i=nfree+1,...,n, index(i) are the indices of bound variables On entry after the first iteration, index gives the free variables at the previous iteration. On exit it gives the free variables based on the determination in cauchy using the array iwhere.
nenter	TODO
ileave	TODO
indx2	On entry indx2 is unspecified. On exit with iter$>$0, indx2 indicates which variables have changed status since the previous iteration. For i= 1,...,nenter, indx2(i) have changed from bound to free. For i= ileave+1,...,n, indx2(i) have changed from free to bound.
iwhere	TODO
wrk	TODO
updatd	TODO
cnstnd	indicating whether bounds are present
iprint	control screen output
iter	TODO

Definition at line 32 of file freev.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.11 src/hpsolb.f File Reference

Functions/Subroutines

- subroutine hpsolb (n, t, iorder, iheap)

  This subroutine sorts out the least element of t, and puts the remaining elements of t in a heap.

3.11.1 Function/Subroutine Documentation

3.11.1.1 hpsolb()

```
subroutine hpsolb (
    integer n,
    double precision, dimension(n) t,
    integer, dimension(n) iorder,
    integer iheap )
```

Parameters

n	On entry n is the dimension of the arrays t and iorder. On exit n is unchanged.
t	On entry t stores the elements to be sorted. On exit t(n) stores the least elements of t, and t(1) to t(n-1) stores the remaining elements in the form of a heap.
iorder	On entry iorder(i) is the index of t(i). On exit iorder(i) is still the index of t(i), but iorder may be permuted in accordance with t.
iheap	On entry iheap should be set as follows: • iheap .eq. 0 if t(1) to t(n) is not in the form of a heap, • iheap .ne. 0 if otherwise. On exit iheap is unchanged.

Definition at line 21 of file hpsolb.f.

Referenced by cauchy().

Here is the caller graph for this function:

3.12 src/lnsrlb.f File Reference

Functions/Subroutines

- subroutine lnsrlb (n, l, u, nbd, x, f, fold, gd, gdold, g, d, r, t, z, stp, dnorm, dtd, xstep, stpmx, iter, ifun, iback, nfgv, info, task, boxed, cnstnd, csave, isave, dsave)

  This subroutine calls subroutine dcsrch from the Minpack2 library to perform the line search. Subroutine dscrch is safeguarded so that all trial points lie within the feasible region.

3.12.1 Function/Subroutine Documentation

3.12.1.1 lnsrlb()

```
subroutine lnsrlb (
        integer n,
        double precision, dimension(n) l,
        double precision, dimension(n) u,
        integer, dimension(n) nbd,
        double precision, dimension(n) x,
        double precision f,
        double precision fold,
        double precision gd,
        double precision gdold,
        double precision, dimension(n) g,
        double precision, dimension(n) d,
        double precision, dimension(n) r,
        double precision, dimension(n) t,
        double precision, dimension(n) z,
        double precision stp,
        double precision dnorm,
        double precision dtd,
        double precision xstep,
        double precision stpmx,
        integer iter,
        integer ifun,
        integer iback,
        integer nfgv,
        integer info,
        character*60 task,
        logical boxed,
        logical cnstnd,
        character*60 csave,
        integer, dimension(2) isave,
        double precision, dimension(13) dsave )
```

Parameters

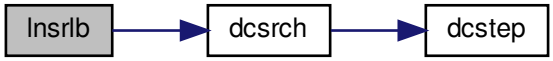
n	number of parameters
l	lower bounds of parameters
u	upper bounds of parameters
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: nbd(i)= • 0 if x(i) is unbounded, • 1 if x(i) has only a lower bound, • 2 if x(i) has both lower and upper bounds, and • 3 if x(i) has only an upper bound. On exit nbd is unchanged.
x	position
f	function value at x
fold	TODO
gd	TODO
gdold	TODO
g	gradient of f at x
d	TODO
r	TODO
t	TODO
z	TODO
stp	TODO
dnorm	TODO
dtd	TODO
xstep	TODO
stpmx	TODO
iter	TODO
ifun	TODO
iback	TODO
nfgv	TODO
info	TODO
task	TODO
boxed	TODO
cnstnd	TODO
csave	working array
isave	working array
dsave	working array

Definition at line 43 of file lnsrlb.f.

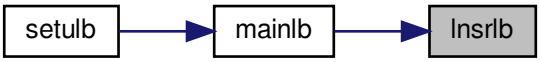
References dcsrch().

Referenced by mainlb().

Here is the call graph for this function:



Here is the caller graph for this function:



3.13 src/mainlb.f File Reference

Functions/Subroutines

- subroutine mainlb (n, m, x, l, u, nbd, f, g, factr, pgtol, ws, wy, sy, ss, wt, wn, snd, z, r, d, t, xp, wa, index, iwhere, indx2, task, iprint, csave, lsave, isave, dsave)

  This subroutine solves bound constrained optimization problems by using the compact formula of the limited memory BFGS updates.

3.13.1 Function/Subroutine Documentation

3.13.1.1 mainlb()

```
subroutine mainlb (
        integer n,
        integer m,
        double precision, dimension(n) x,
        double precision, dimension(n) l,
        double precision, dimension(n) u,
        integer, dimension(n) nbd,
        double precision f,
        double precision, dimension(n) g,
        double precision factr,
        double precision pgtol,
        double precision, dimension(n, m) ws,
        double precision, dimension(n, m) wy,
        double precision, dimension(m, m) sy,
        double precision, dimension(m, m) ss,
```

```
double precision, dimension(m, m) wt,
double precision, dimension(2*m, 2*m) wn,
double precision, dimension(2*m, 2*m) snd,
double precision, dimension(n) z,
double precision, dimension(n) r,
double precision, dimension(n) d,
double precision, dimension(n) t,
double precision, dimension(n) xp,
double precision, dimension(8*m) wa,
integer, dimension(n) index,
integer, dimension(n) iwhere,
integer, dimension(n) indx2,
character*60 task,
integer iprint,
character*60 csave,
logical, dimension(4) lsave,
integer, dimension(23) isave,
double precision, dimension(29) dsave )
```

This subroutine solves bound constrained optimization problems by using the compact formula of the limited memory BFGS updates.

Parameters

n	On entry n is the number of variables. On exit n is unchanged.
m	On entry m is the maximum number of variable metric corrections allowed in the limited memory matrix. On exit m is unchanged.
x	On entry x is an approximation to the solution. On exit x is the current approximation.
l	On entry l is the lower bound of x. On exit l is unchanged.
u	On entry u is the upper bound of x. On exit u is unchanged.
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: 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. On exit nbd is unchanged.
f	On first entry f is unspecified. On final exit f is the value of the function at x.
g	On first entry g is unspecified. On final exit g is the value of the gradient at x.
factr	On entry factr >= 0 is specified by the user. The iteration will stop when (f^k - f^{k+1})/max{|f^k|,|f^{k+1}|,1} <= factr*epsmch where epsmch is the machine precision, which is automatically generated by the code. On exit factr is unchanged.
pgtol	On entry pgtol >= 0 is specified by the user. The iteration will stop when max{|proj g_i | i = 1, ..., n} <= pgtol where pg_i is the ith component of the projected gradient. On exit pgtol is unchanged.

Parameters

ws	On entry this stores S, a set of s-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
wy	On entry this stores Y, a set of y-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
sy	On entry this stores S'Y, that defines the limited memory BFGS matrix. On exit this array is unchanged.
ss	On entry this stores S'S, that defines the limited memory BFGS matrix. On exit this array is unchanged.
wt	On entry this stores the Cholesky factorization of (theta∗S'S+LD^(-1)L'), that defines the limited memory BFGS matrix. See eq. (2.26) in [3]. On exit this array is unchanged.
wn	working array used to store the LEL^T factorization of the indefinite matrix K = [-D -Y'ZZ'Y/theta L_a'-R_z'] [L_a -R_z theta∗S'AA'S] where E = [-I 0] [0 I]
snd	working array used to store the lower triangular part of N = [Y' ZZ'Y L_a'+R_z'] [L_a +R_z S'AA'S]
z	working array used at different times to store the Cauchy point and the Newton point.
r	working array
d	working array
t	working array
xp	working array used to safeguard the projected Newton direction
wa	working array
index	In subroutine freev, index is used to store the free and fixed variables at the Generalized Cauchy Point (GCP).
iwhere	working array used to record the status of the vector x for GCP computation. iwhere(i)= • 0 or -3 if x(i) is free and has bounds, • 1 if x(i) is fixed at l(i), and l(i) .ne. u(i) • 2 if x(i) is fixed at u(i), and u(i) .ne. l(i) • 3 if x(i) is always fixed, i.e., u(i)=x(i)=l(i) • -1 if x(i) is always free, i.e., no bounds on it.
indx2	working array Within subroutine cauchy, indx2 corresponds to the array iorder. In subroutine freev, a list of variables entering and leaving the free set is stored in indx2, and it is passed on to subroutine formk with this information.
task	working string indicating the current job when entering and leaving this subroutine.
iprint	It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.
csave	working string

Parameters

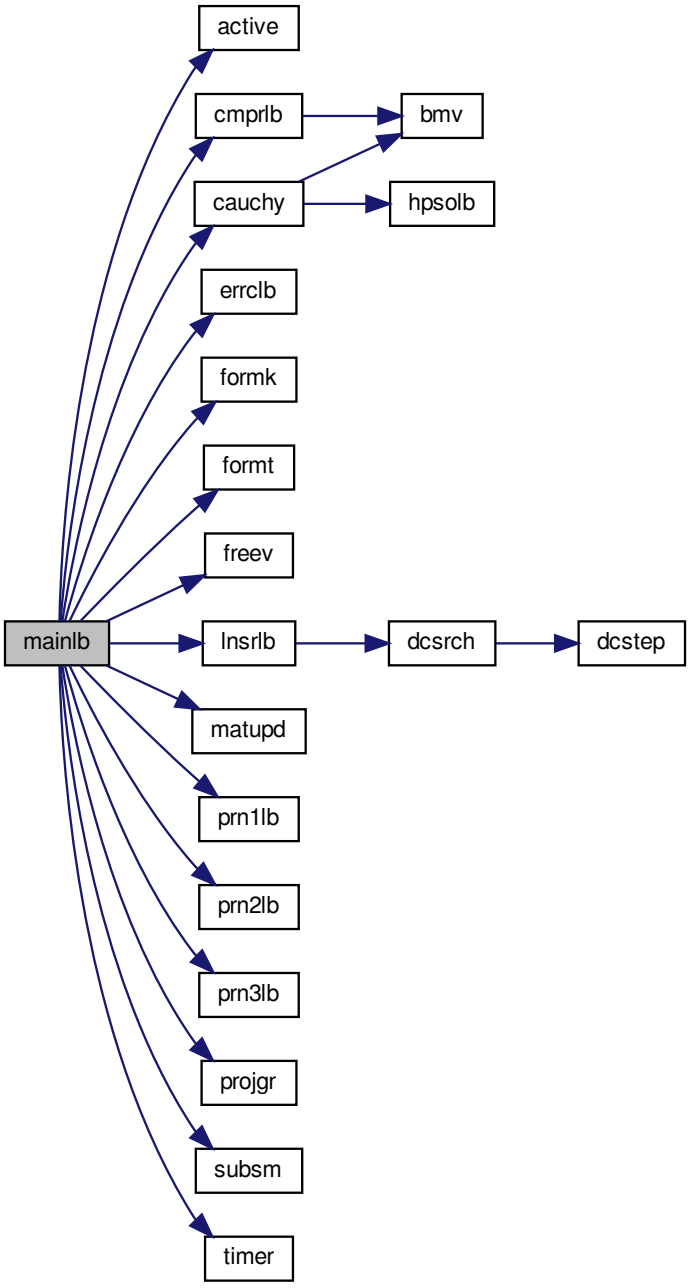
lsave	working array
isave	working array
dsave	working array

Definition at line 128 of file mainlb.f.

References active(), cauchy(), cmprlb(), errclb(), formk(), formt(), freev(), lnsrlb(), matupd(), prn1lb(), prn2lb(), prn3lb(), projgr(), subsm(), and timer().

Referenced by setulb().

Here is the call graph for this function:



Here is the caller graph for this function:

3.14 src/matupd.f File Reference

Functions/Subroutines

- subroutine matupd (n, m, ws, wy, sy, ss, d, r, itail, iupdat, col, head, theta, rr, dr, stp, dtd)

  This subroutine updates matrices WS and WY, and forms the middle matrix in B.

3.14.1 Function/Subroutine Documentation

3.14.1.1 matupd() subroutine matupd (

```
integer n,
integer m,
double precision, dimension(n, m) ws,
double precision, dimension(n, m) wy,
double precision, dimension(m, m) sy,
double precision, dimension(m, m) ss,
double precision, dimension(n) d,
double precision, dimension(n) r,
integer itail,
integer iupdat,
integer col,
integer head,
double precision theta,
double precision rr,
double precision dr,
double precision stp,
double precision dtd )
```

This subroutine updates matrices WS and WY, and forms the middle matrix in B.

Parameters

n	On entry n is the number of variables. On exit n is unchanged.
m	On entry m is the maximum number of variable metric corrections allowed in the limited memory matrix. On exit m is unchanged.
ws	On entry this stores S, a set of s-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.

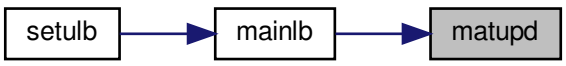
Parameters

wy	On entry this stores Y, a set of y-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
sy	On entry this stores S'Y, that defines the limited memory BFGS matrix. On exit this array is unchanged.
ss	On entry this stores S'S, that defines the limited memory BFGS matrix. On exit this array is unchanged.
d	TODO
r	TODO
itail	TODO
iupdat	TODO
col	On entry col is the actual number of variable metric corrections stored so far. On exit col is unchanged.
head	On entry head is the location of the first s-vector (or y-vector) in S (or Y). On exit col is unchanged.
theta	On entry theta is the scaling factor specifying B_0 = theta I. On exit theta is unchanged.
rr	TODO
dr	TODO
stp	TODO
dtd	TODO

Definition at line 49 of file matupd.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.15 src/prn1lb.f File Reference

Functions/Subroutines

- subroutine prn1lb (n, m, l, u, x, iprint, itfile, epsmch)

  This subroutine prints the input data, initial point, upper and lower bounds of each variable, machine precision, as well as the headings of the output.

3.15.1 Function/Subroutine Documentation

3.15.1.1 prn1lb()

```
subroutine prn1lb (
    integer n,
    integer m,
    double precision, dimension(n) l,
    double precision, dimension(n) u,
    double precision, dimension(n) x,
    integer iprint,
    integer itfile,
    double precision epsmch )
```

This subroutine prints the input data, initial point, upper and lower bounds of each variable, machine precision, as well as the headings of the output.

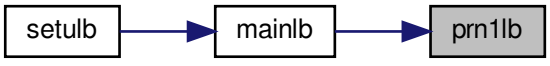
Parameters

n	On entry n is the number of variables. On exit n is unchanged.
m	On entry m is the maximum number of variable metric corrections allowed in the limited memory matrix. On exit m is unchanged.
l	On entry l is the lower bound of x. On exit l is unchanged.
u	On entry u is the upper bound of x. On exit u is unchanged.
x	On entry x is an approximation to the solution. On exit x is the current approximation.
iprint	It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.
itfile	unit number of iterate.dat file
epsmch	machine precision epsilon

Definition at line 39 of file prn1lb.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.16 src/prn2lb.f File Reference

Functions/Subroutines

- subroutine prn2lb (n, x, f, g, iprint, itfile, iter, nfgv, nact, sbgnrm, nseg, word, iword, iback, stp, xstep)

  This subroutine prints out new information after a successful line search.

3.16.1 Function/Subroutine Documentation

3.16.1.1 prn2lb()

```
subroutine prn2lb (
    integer n,
    double precision, dimension(n) x,
    double precision f,
    double precision, dimension(n) g,
    integer iprint,
    integer itfile,
    integer iter,
    integer nfgv,
    integer nact,
    double precision sbgnrm,
    integer nseg,
    character*3 word,
    integer iword,
    integer iback,
    double precision stp,
    double precision xstep )
```

This subroutine prints out new information after a successful line search.

Parameters

n	On entry n is the number of variables. On exit n is unchanged.
x	On entry x is an approximation to the solution. On exit x is the current approximation.
f	On first entry f is unspecified. On final exit f is the value of the function at x.
g	On first entry g is unspecified. On final exit g is the value of the gradient at x.
iprint	It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.

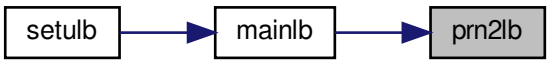
Parameters

itfile	unit number of iterate.dat file
iter	TODO
nfgv	TODO
nact	TODO
sbgnrm	TODO
nseg	TODO
word	TODO
iword	TODO
iback	TODO
stp	TODO
xstep	TODO

Definition at line 43 of file prn2lb.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.17 src/prn3lb.f File Reference

Functions/Subroutines

- subroutine prn3lb (n, x, f, task, iprint, info, itfile, iter, nfgv, nintol, nskip, nact, sbgnrm, time, nseg, word, iback, stp, xstep, k, cachyt, sbtime, lnscht)

  This subroutine prints out information when either a built-in convergence test is satisfied or when an error message is generated.

3.17.1 Function/Subroutine Documentation

3.17.1.1 prn3lb()

```
subroutine prn3lb (
    integer n,
    double precision, dimension(n) x,
    double precision f,
    character*60 task,
    integer iprint,
    integer info,
    integer itfile,
    integer iter,
    integer nfgv,
    integer nintol,
    integer nskip,
    integer nact,
    double precision sbgnrm,
    double precision time,
    integer nseg,
    character*3 word,
    integer iback,
    double precision stp,
    double precision xstep,
    integer k,
    double precision cachyt,
    double precision sbtime,
    double precision lnscht )
```

This subroutine prints out information when either a built-in convergence test is satisfied or when an error message is generated.

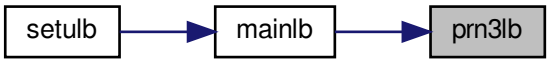
Parameters

n	On entry n is the number of variables. On exit n is unchanged.
x	On entry x is an approximation to the solution. On exit x is the current approximation.
f	On first entry f is unspecified. On final exit f is the value of the function at x.
task	working string indicating the current job when entering and leaving this subroutine.
iprint	TODO
info	TODO
itfile	unit number of iterate.dat file
iter	TODO
nfgv	TODO
nintol	TODO
nskip	TODO
nact	TODO
sbgnrm	TODO
time	TODO
nseg	TODO
word	TODO
iback	TODO
stp	TODO
xstep	TODO
k	TODO
cachyt	TODO
sbtime	TODO
lnscht	TODO

Definition at line 41 of file prn3lb.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.18 src/projgr.f File Reference

Functions/Subroutines

- subroutine projgr (n, l, u, nbd, x, g, sbgnrm)

  This subroutine computes the infinity norm of the projected gradient.

3.18.1 Function/Subroutine Documentation

3.18.1.1 projgr()

```
subroutine projgr (
    integer n,
    double precision, dimension(n) l,
    double precision, dimension(n) u,
    integer, dimension(n) nbd,
    double precision, dimension(n) x,
    double precision, dimension(n) g,
    double precision sbgnrm )
```

This subroutine computes the infinity norm of the projected gradient.

Parameters

n	On entry n is the number of variables. On exit n is unchanged.
l	On entry l is the lower bound of x. On exit l is unchanged.
u	On entry u is the upper bound of x. On exit u is unchanged.
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: 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. On exit nbd is unchanged.

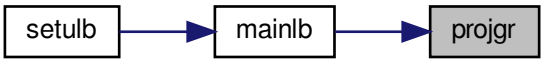
Parameters

x	On entry x is an approximation to the solution. On exit x is unchanged.
g	On entry g is the gradient. On exit g is unchanged.
sbgnrm	infinity norm of projected gradient

Definition at line 33 of file projgr.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.19 src/setulb.f File Reference

Functions/Subroutines

- subroutine setulb (n, m, x, l, u, nbd, f, g, factr, pgtol, wa, iwa, task, iprint, csave, lsave, isave, dsave)

  This subroutine partitions the working arrays wa and iwa, and then uses the limited memory BFGS method to solve the bound constrained optimization problem by calling mainlb.

3.19.1 Function/Subroutine Documentation

3.19.1.1 setulb()

```
subroutine setulb (
        integer n,
        integer m,
        double precision, dimension(n) x,
        double precision, dimension(n) l,
        double precision, dimension(n) u,
        integer, dimension(n) nbd,
        double precision f,
        double precision, dimension(n) g,
        double precision factr,
        double precision pgtol,
        double precision, dimension(2*m*n + 5*n + 11*m*m + 8*m) wa,
        integer, dimension(3*n) iwa,
        character*60 task,
        integer iprint,
```

```
character*60 csave,
logical, dimension(4) lsave,
integer, dimension(44) isave,
double precision, dimension(29) dsave )
```

This subroutine partitions the working arrays wa and iwa, and then uses the limited memory BFGS method to solve the bound constrained optimization problem by calling mainlb. (The direct method will be used in the subspace minimization.)

Parameters

n	On entry n is the dimension of the problem. On exit n is unchanged.
m	On entry m is the maximum number of variable metric corrections used to define the limited memory matrix. On exit m is unchanged.
x	On entry x is an approximation to the solution. On exit x is the current approximation.
l	On entry l is the lower bound on x. On exit l is unchanged.
u	On entry u is the upper bound on x. On exit u is unchanged.
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: 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, and • 3 if x(i) has only an upper bound. On exit nbd is unchanged.
f	On first entry f is unspecified. On final exit f is the value of the function at x.
g	On first entry g is unspecified. On final exit g is the value of the gradient at x.
factr	On entry factr >= 0 is specified by the user. The iteration will stop when (f^k - f^{k+1})/max{|f^k|,|f^{k+1}|,1} <= factr*epsmch where epsmch is the machine precision, which is automatically generated by the code. Typical values for factr: • 1.d+12 for low accuracy; • 1.d+7 for moderate accuracy; • 1.d+1 for extremely high accuracy. On exit factr is unchanged.
pgtol	On entry pgtol >= 0 is specified by the user. The iteration will stop when max{|proj g_i | i = 1, ..., n} <= pgtol where pg_i is the ith component of the projected gradient. On exit pgtol is unchanged.
wa	working array
iwa	working array
task	working string indicating the current job when entering and quitting this subroutine.

Parameters

iprint	Must be set by the user. It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.
csave	working string
lsave	working array; On exit with 'task' = NEW_X, the following information is available: • If lsave(1) = .true. then the initial X has been replaced by its projection in the feasible set; • If lsave(2) = .true. then the problem is constrained; • If lsave(3) = .true. then each variable has upper and lower bounds;
isave	working array; On exit with 'task' = NEW_X, the following information is available: • isave(22) = the total number of intervals explored in the search of Cauchy points; • isave(26) = the total number of skipped BFGS updates before the current iteration; • isave(30) = the number of current iteration; • isave(31) = the total number of BFGS updates prior the current iteration; • isave(33) = the number of intervals explored in the search of Cauchy point in the current iteration; • isave(34) = the total number of function and gradient evaluations; • isave(36) = the number of function value or gradient evaluations in the current iteration; • if isave(37) = 0 then the subspace argmin is within the box; • if isave(37) = 1 then the subspace argmin is beyond the box; • isave(38) = the number of free variables in the current iteration; • isave(39) = the number of active constraints in the current iteration; • n + 1 - isave(40) = the number of variables leaving the set of active constraints in the current iteration; • isave(41) = the number of variables entering the set of active constraints in the current iteration.

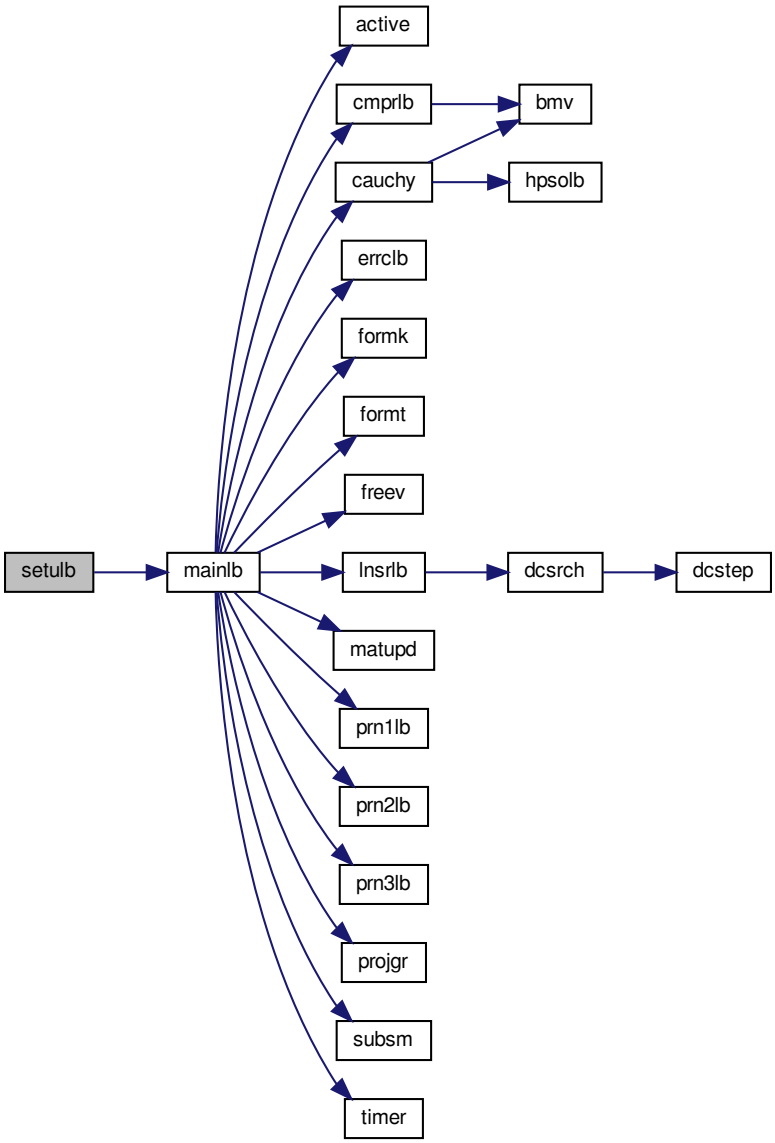
Parameters

dsave	working array; On exit with 'task' = NEW_X, the following information is available: • dsave(1) = current 'theta' in the BFGS matrix; • dsave(2) = f(x) in the previous iteration; • dsave(3) = factr∗epsmch; • dsave(4) = 2-norm of the line search direction vector; • dsave(5) = the machine precision epsmch generated by the code; • dsave(7) = the accumulated time spent on searching for Cauchy points; • dsave(8) = the accumulated time spent on subspace minimization; • dsave(9) = the accumulated time spent on line search; • dsave(11) = the slope of the line search function at the current point of line search; • dsave(12) = the maximum relative step length imposed in line search; • dsave(13) = the infinity norm of the projected gradient; • dsave(14) = the relative step length in the line search; • dsave(15) = the slope of the line search function at the starting point of the line search; • dsave(16) = the square of the 2-norm of the line search direction vector.

Definition at line 188 of file setulb.f.

References mainlb().

Here is the call graph for this function:



3.20 src/subsm.f File Reference

Functions/Subroutines

- subroutine subsm (n, m, nsub, ind, l, u, nbd, x, d, xp, ws, wy, theta, xx, gg, col, head, iword, wv, wn, iprint, info)

  Performs the subspace minimization.

3.20.1 Function/Subroutine Documentation

3.20.1.1 subsm()

```
subroutine subsm (
        integer n,
        integer m,
        integer nsub,
        integer, dimension(nsub) ind,
        double precision, dimension(n) l,
        double precision, dimension(n) u,
        integer, dimension(n) nbd,
        double precision, dimension(n) x,
        double precision, dimension(n) d,
        double precision, dimension(n) xp,
        double precision, dimension(n, m) ws,
        double precision, dimension(n, m) wy,
        double precision theta,
        double precision, dimension(n) xx,
        double precision, dimension(n) gg,
        integer col,
        integer head,
        integer iword,
        double precision, dimension(2*m) wv,
        double precision, dimension(2*m, 2*m) wn,
        integer iprint,
        integer info )
```

Given xcp, l, u, r, an index set that specifies the active set at xcp, and an l-BFGS matrix B (in terms of WY, WS, SY, WT, head, col, and theta), this subroutine computes an approximate solution of the subspace problem

(P) min Q(x) = r'(x-xcp) + 1/2 (x-xcp)' B (x-xcp)

subject to l<=x<=u x_i=xcp_i for all i in A(xcp)

along the subspace unconstrained Newton direction

d = -(Z'BZ)^(-1) r.

The formula for the Newton direction, given the L-BFGS matrix and the Sherman-Morrison formula, is

d = (1/theta)r + (1/theta*2) Z'WK^(-1)W'Z r.

where K = [-D -Y'ZZ'Y/theta L_a'-R_z'] [L_a -R_z theta*S'AA'S]

Note that this procedure for computing d differs from that described in [1]. One can show that the matrix K is equal to the matrix M^[-1]N in that paper.

Parameters

n	On entry n is the dimension of the problem. On exit n is unchanged.
m	On entry m is the maximum number of variable metric corrections used to define the limited memory matrix. On exit m is unchanged.
nsub	On entry nsub is the number of free variables. On exit nsub is unchanged.
ind	On entry ind specifies the coordinate indices of free variables. On exit ind is unchanged.
l	On entry l is the lower bound of x. On exit l is unchanged.

Parameters

u	On entry u is the upper bound of x. On exit u is unchanged.
nbd	On entry nbd represents the type of bounds imposed on the variables, and must be specified as follows: 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, and • 3 if x(i) has only an upper bound. On exit nbd is unchanged.
x	On entry x specifies the Cauchy point xcp. On exit x(i) is the minimizer of Q over the subspace of free variables.
d	On entry d is the reduced gradient of Q at xcp. On exit d is the Newton direction of Q.
xp	used to safeguard the projected Newton direction
xx	On entry it holds the current iterate. On output it is unchanged.
gg	On entry it holds the gradient at the current iterate. On output it is unchanged.
ws	On entry this stores S, a set of s-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
wy	On entry this stores Y, a set of y-vectors, that defines the limited memory BFGS matrix. On exit this array is unchanged.
theta	On entry theta is the scaling factor specifying B_0 = theta I. On exit theta is unchanged.
col	On entry col is the actual number of variable metric corrections stored so far. On exit col is unchanged.
head	On entry head is the location of the first s-vector (or y-vector) in S (or Y). On exit col is unchanged.
iword	On entry iword is unspecified. On exit iword specifies the status of the subspace solution. iword = • 0 if the solution is in the box, • 1 if some bound is encountered.
wv	working array
wn	On entry the upper triangle of wn stores the LEL^T factorization of the indefinite matrix K = [-D -Y'ZZ'Y/theta L_a'-R_z'] [L_a -R_z theta∗S'AA'S] where E = [-I 0] [0 I] On exit wn is unchanged.

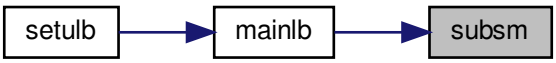
Parameters

iprint	must be set by the user; It controls the frequency and type of output generated: • iprint<0 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 n-vectors; • iprint=100 print also the changes of active set and final x; • iprint>100 print details of every iteration including x and g; When iprint > 0, the file iterate.dat will be created to summarize the iteration.
info	On entry info is unspecified. On exit info = • 0 for normal return, • nonzero for abnormal return when the matrix K is ill-conditioned.

Definition at line 124 of file subsm.f.

Referenced by mainlb().

Here is the caller graph for this function:



3.21 src/timer.f File Reference

Functions/Subroutines

- subroutine timer (ttime)

  This routine computes cpu time in double precision.

3.21.1 Function/Subroutine Documentation

3.21.1.1 timer()

```
subroutine timer (
        double precision ttime )
```

This routine computes cpu time in double precision; it makes use of the intrinsic f90 cpu_time therefore a conversion type is needed.

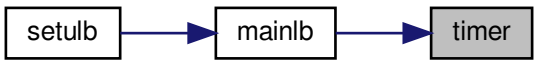
Parameters

ttime	CPU time in double precision

Definition at line 10 of file timer.f.

Referenced by mainlb().

Here is the caller graph for this function:



4 Example Documentation

4.1 driver1.f

This simple driver demonstrates how to call the L-BFGS-B code to solve a sample problem (the extended Rosenbrock function subject to bounds on the variables). The dimension n of this problem is variable. (Fortran-77 version)

```
1 c> \file driver1.f
2
3 c
4 c  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 c  or "3-clause license")
6 c  Please read attached file License.txt
7 c
8 c                           DRIVER 1 in Fortran 77
9 c     ----------------------------------------------------------------
10 c               SIMPLE DRIVER FOR L-BFGS-B (version 3.0)
11 c     ----------------------------------------------------------------
12 c
13 c        L-BFGS-B is a code for solving large nonlinear optimization
14 c             problems with simple bounds on the variables.
15 c
16 c        The code can also be used for unconstrained problems and is
17 c        as efficient for these problems as the earlier limited memory
18 c                         code L-BFGS.
19 c
20 c        This is the simplest driver in the package. It uses all the
21 c                    default settings of the code.
22 c
23 c
24 c     References:
25 c
26 c        [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
27 c        memory algorithm for bound constrained optimization",
28 c        SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
29 c
30 c        [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
31 c        Subroutines for Large Scale Bound Constrained Optimization"
32 c        Tech. Report, NAM-11, EECS Department, Northwestern University,
33 c        1994.
34 c
35 c
36 c          (Postscript files of these papers are available via anonymous
37 c           ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
38 c
39 c                               *  *  *
40 c
41 c         March 2011   (latest revision)
```

```
c          Optimization Center at Northwestern University
c          Instituto Tecnologico Autonomo de Mexico
c
c          Jorge Nocedal and Jose Luis Morales, Remark on "Algorithm 778:
c          L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained
c          Optimization"  (2011). To appear in  ACM Transactions on
c          Mathematical Software,
c
c     ============================================================
c             DESCRIPTION OF THE VARIABLES IN L-BFGS-B
c     ============================================================
c
c     n is an INTEGER variable that must be set by the user to the
c       number of variables.  It is not altered by the routine.
c
c     m is an INTEGER variable that must be set by the user to the
c       number of corrections used in the limited memory matrix.
c       It is not altered by the routine.  Values of m < 3  are
c       not recommended, and large values of m can result in excessive
c       computing time. The range  3 <= m <= 20 is recommended.
c
c     x is a DOUBLE PRECISION array of length n.  On initial entry
c       it must be set by the user to the values of the initial
c       estimate of the solution vector.  Upon successful exit, it
c       contains the values of the variables at the best point
c       found (usually an approximate solution).
c
c     l is a DOUBLE PRECISION array of length n that must be set by
c       the user to the values of the lower bounds on the variables. If
c       the i-th variable has no lower bound, l(i) need not be defined.
c
c     u is a DOUBLE PRECISION array of length n that must be set by
c       the user to the values of the upper bounds on the variables. If
c       the i-th variable has no upper bound, u(i) need not be defined.
c
c     nbd is an INTEGER array of dimension n that must be set by the
c       user to the type of bounds imposed on the variables:
c       nbd(i)=0 if x(i) is unbounded,
c              1 if x(i) has only a lower bound,
c              2 if x(i) has both lower and upper bounds,
c              3 if x(i) has only an upper bound.
c
c     f is a DOUBLE PRECISION variable.  If the routine setulb returns
c       with task(1:2)= 'FG', then f must be set by the user to
c       contain the value of the function at the point x.
c
c     g is a DOUBLE PRECISION array of length n.  If the routine setulb
c       returns with taskb(1:2)= 'FG', then g must be set by the user to
c       contain the components of the gradient at the point x.
c
c     factr is a DOUBLE PRECISION variable that must be set by the user.
c       It is a tolerance in the termination test for the algorithm.
c       The iteration will stop when
c
c         (f^k - f^{k+1})/max{|f^k|,|f^{k+1}|,1} <= factr*epsmch
c
c       where epsmch is the machine precision which is automatically
c       generated by the code. Typical values for factr on a computer
c        with 15 digits of accuracy in double precision are:
c        factr=1.d+12 for low accuracy;
c              1.d+7  for moderate accuracy;
c              1.d+1  for extremely high accuracy.
c       The user can suppress this termination test by setting factr=0.
c
c     pgtol is a double precision variable.
c       On entry pgtol >= 0 is specified by the user.  The iteration
c         will stop when
c
c                 max{|proj g_i | i = 1, ..., n} <= pgtol
c
c         where pg_i is the ith component of the projected gradient.
c       The user can suppress this termination test by setting pgtol=0.
c
c     wa is a DOUBLE PRECISION  array of length
c       (2mmax + 5)nmax + 11mmax^2 + 8mmax used as workspace.
c       This array must not be altered by the user.
c
c     iwa is an INTEGER  array of length 3nmax used as
c       workspace. This array must not be altered by the user.
c
c     task is a CHARACTER string of length 60.
c       On first entry, it must be set to 'START'.
c       On a return with task(1:2)='FG', the user must evaluate the
c         function f and gradient g at the returned value of x.
c       On a return with task(1:5)='NEW_X', an iteration of the
c         algorithm has concluded, and f and g contain f(x) and g(x)
c         respectively.  The user can decide whether to continue or stop
```

```
129 c         the iteration.
130 c       When
131 c         task(1:4)='CONV', the termination test in L-BFGS-B has been
132 c           satisfied;
133 c         task(1:4)='ABNO', the routine has terminated abnormally
134 c           without being able to satisfy the termination conditions,
135 c           x contains the best approximation found,
136 c           f and g contain f(x) and g(x) respectively;
137 c         task(1:5)='ERROR', the routine has detected an error in the
138 c           input parameters;
139 c       On exit with task = 'CONV', 'ABNO' or 'ERROR', the variable task
140 c         contains additional information that the user can print.
141 c       This array should not be altered unless the user wants to
142 c          stop the run for some reason.  See driver2 or driver3
143 c          for a detailed explanation on how to stop the run
144 c          by assigning task(1:4)='STOP' in the driver.
145 c
146 c     iprint is an INTEGER variable that must be set by the user.
147 c       It controls the frequency and type of output generated:
148 c        iprint<0    no output is generated;
149 c        iprint=0    print only one line at the last iteration;
150 c        0<iprint<99 print also f and |proj g| every iprint iterations;
151 c        iprint=99   print details of every iteration except n-vectors;
152 c        iprint=100  print also the changes of active set and final x;
153 c        iprint>100  print details of every iteration including x and g;
154 c       When iprint > 0, the file iterate.dat will be created to
155 c                        summarize the iteration.
156 c
157 c     csave  is a CHARACTER working array of length 60.
158 c
159 c     lsave is a LOGICAL working array of dimension 4.
160 c       On exit with task = 'NEW_X', the following information is
161 c         available:
162 c       lsave(1) = .true.  the initial x did not satisfy the bounds;
163 c       lsave(2) = .true.  the problem contains bounds;
164 c       lsave(3) = .true.  each variable has upper and lower bounds.
165 c
166 c     isave is an INTEGER working array of dimension 44.
167 c       On exit with task = 'NEW_X', it contains information that
168 c       the user may want to access:
169 c         isave(30) = the current iteration number;
170 c         isave(34) = the total number of function and gradient
171 c                         evaluations;
172 c         isave(36) = the number of function value or gradient
173 c                                  evaluations in the current iteration;
174 c         isave(38) = the number of free variables in the current
175 c                         iteration;
176 c         isave(39) = the number of active constraints at the current
177 c                         iteration;
178 c
179 c         see the subroutine setulb.f for a description of other
180 c         information contained in isave
181 c
182 c     dsave is a DOUBLE PRECISION working array of dimension 29.
183 c       On exit with task = 'NEW_X', it contains information that
184 c         the user may want to access:
185 c         dsave(2) = the value of f at the previous iteration;
186 c         dsave(5) = the machine precision epsmch generated by the code;
187 c         dsave(13) = the infinity norm of the projected gradient;
188 c
189 c         see the subroutine setulb.f for a description of other
190 c         information contained in dsave
191 c
192 c     ------------------------------------------------------------
193 c           END OF THE DESCRIPTION OF THE VARIABLES IN L-BFGS-B
194 c     ------------------------------------------------------------
195
196       program driver
197
198 c     This simple driver demonstrates how to call the L-BFGS-B code to
199 c       solve a sample problem (the extended Rosenbrock function
200 c       subject to bounds on the variables). The dimension n of this
201 c       problem is variable.
202
203       integer          nmax, mmax
204       parameter(nmax=1024, mmax=17)
205 c        nmax is the dimension of the largest problem to be solved.
206 c        mmax is the maximum number of limited memory corrections.
207
208 c     Declare the variables needed by the code.
209 c       A description of all these variables is given at the end of
210 c       the driver.
211
212       character*60     task, csave
213       logical          lsave(4)
214       integer          n, m, iprint,
215      +                 nbd(nmax), iwa(3*nmax), isave(44)
```

```
216       double precision f, factr, pgtol,
217      +                 x(nmax), l(nmax), u(nmax), g(nmax), dsave(29),
218      +                 wa(2*mmax*nmax + 5*nmax + 11*mmax*mmax + 8*mmax)
219
220 c     Declare a few additional variables for this sample problem.
221
222       double precision t1, t2
223       integer          i
224
225 c     We wish to have output at every iteration.
226
227       iprint = 1
228
229 c     We specify the tolerances in the stopping criteria.
230
231       factr=1.0d+7
232       pgtol=1.0d-5
233
234 c     We specify the dimension n of the sample problem and the number
235 c        m of limited memory corrections stored.  (n and m should not
236 c        exceed the limits nmax and mmax respectively.)
237
238       n=25
239       m=5
240
241 c     We now provide nbd which defines the bounds on the variables:
242 c                     l   specifies the lower bounds,
243 c                     u   specifies the upper bounds.
244
245 c     First set bounds on the odd-numbered variables.
246
247       do 10 i=1,n,2
248          nbd(i)=2
249          l(i)=1.0d0
250          u(i)=1.0d2
251    10 continue
252
253 c     Next set bounds on the even-numbered variables.
254
255       do 12 i=2,n,2
256          nbd(i)=2
257          l(i)=-1.0d2
258          u(i)=1.0d2
259    12  continue
260
261 c     We now define the starting point.
262
263       do 14 i=1,n
264          x(i)=3.0d0
265    14 continue
266
267       write (6,16)
268    16 format(/,5x, 'Solving sample problem.',
269      +       /,5x, ' (f = 0.0 at the optimal solution.)',/)
270
271 c     We start the iteration by initializing task.
272 c
273       task = 'START'
274
275 c        ------- the beginning of the loop ----------
276
277  111  continue
278
279 c     This is the call to the L-BFGS-B code.
280
281       call setulb(n,m,x,l,u,nbd,f,g,factr,pgtol,wa,iwa,task,iprint,
282      +            csave,lsave,isave,dsave)
283
284       if (task(1:2) .eq. 'FG') then
285 c        the minimization routine has returned to request the
286 c        function f and gradient g values at the current x.
287
288 c        Compute function value f for the sample problem.
289
290          f=.25d0*(x(1)-1.d0)**2
291          do 20 i=2,n
292             f=f+(x(i)-x(i-1)**2)**2
293    20    continue
294          f=4.d0*f
295
296 c        Compute gradient g for the sample problem.
297
298          t1=x(2)-x(1)**2
299          g(1)=2.d0*(x(1)-1.d0)-1.6d1*x(1)*t1
300          do 22 i=2,n-1
301             t2=t1
302             t1=x(i+1)-x(i)**2
```

```
303             g(i)=8.d0*t2-1.6d1*x(i)*t1
304    22     continue
305            g(n)=8.d0*t1
306
307 c          go back to the minimization routine.
308            goto 111
309         endif
310 c
311         if (task(1:5) .eq. 'NEW_X')  goto 111
312 c          the minimization routine has returned with a new iterate,
313 c           and we have opted to continue the iteration.
314
315 c             ---------- the end of the loop -------------
316
317 c       If task is neither FG nor NEW_X we terminate execution.
318
319         stop
320
321         end
322
323 c======================= The end of driver1 ============================
```

4.2 driver1.f90

This simple driver demonstrates how to call the L-BFGS-B code to solve a sample problem (the extended Rosenbrock function subject to bounds on the variables). The dimension n of this problem is variable. (Fortran-90 version)

```
1 !> \file driver1.f90
2
3 !
4 !  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 !  or "3-clause license")
6 !  Please read attached file License.txt
7 !
8 !
9 !                     DRIVER1 in Fortran 90
10 !     ----------------------------------------------------------------
11 !
12 !        L-BFGS-B is a code for solving large nonlinear optimization
13 !             problems with simple bounds on the variables.
14 !
15 !        The code can also be used for unconstrained problems and is
16 !        as efficient for these problems as the earlier limited memory
17 !                          code L-BFGS.
18 !
19 !        This is the simplest driver in the package. It uses all the
20 !                    default settings of the code.
21 !
22 !
23 !     References:
24 !
25 !        [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
26 !        memory algorithm for bound constrained optimization",
27 !        SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
28 !
29 !        [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
30 !        Subroutines for Large Scale Bound Constrained Optimization"
31 !        Tech. Report, NAM-11, EECS Department, Northwestern University,
32 !        1994.
33 !
34 !
35 !          (Postscript files of these papers are available via anonymous
36 !           ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
37 !
38 !                                * *  *
39 !
40 !         March 2011   (latest revision)
41 !         Optimization Center at Northwestern University
42 !         Instituto Tecnologico Autonomo de Mexico
43 !
44 !         Jorge Nocedal and Jose Luis Morales
45 !
46 !     ----------------------------------------------------------------
47 !             DESCRIPTION OF THE VARIABLES IN L-BFGS-B
48 !     ----------------------------------------------------------------
49 !
50 !     n is an INTEGER variable that must be set by the user to the
51 !       number of variables.  It is not altered by the routine.
52 !
53 !     m is an INTEGER variable that must be set by the user to the
54 !       number of corrections used in the limited memory matrix.
55 !       It is not altered by the routine.  Values of m < 3  are
56 !       not recommended, and large values of m can result in excessive
```

```
57 !       computing time. The range  3 <= m <= 20 is recommended.
58 !
59 !     x is a DOUBLE PRECISION array of length n.  On initial entry
60 !       it must be set by the user to the values of the initial
61 !       estimate of the solution vector.  Upon successful exit, it
62 !       contains the values of the variables at the best point
63 !       found (usually an approximate solution).
64 !
65 !     l is a DOUBLE PRECISION array of length n that must be set by
66 !       the user to the values of the lower bounds on the variables. If
67 !       the i-th variable has no lower bound, l(i) need not be defined.
68 !
69 !     u is a DOUBLE PRECISION array of length n that must be set by
70 !       the user to the values of the upper bounds on the variables. If
71 !       the i-th variable has no upper bound, u(i) need not be defined.
72 !
73 !     nbd is an INTEGER array of dimension n that must be set by the
74 !       user to the type of bounds imposed on the variables:
75 !       nbd(i)=0 if x(i) is unbounded,
76 !              1 if x(i) has only a lower bound,
77 !              2 if x(i) has both lower and upper bounds,
78 !              3 if x(i) has only an upper bound.
79 !
80 !     f is a DOUBLE PRECISION variable.  If the routine setulb returns
81 !       with task(1:2)= 'FG', then f must be set by the user to
82 !       contain the value of the function at the point x.
83 !
84 !     g is a DOUBLE PRECISION array of length n.  If the routine setulb
85 !       returns with taskb(1:2)= 'FG', then g must be set by the user to
86 !       contain the components of the gradient at the point x.
87 !
88 !     factr is a DOUBLE PRECISION variable that must be set by the user.
89 !       It is a tolerance in the termination test for the algorithm.
90 !       The iteration will stop when
91 !
92 !        (f^k - f^{k+1})/max{|f^k|,|f^{k+1}|,1} <= factr*epsmch
93 !
94 !       where epsmch is the machine precision which is automatically
95 !       generated by the code. Typical values for factr on a computer
96 !       with 15 digits of accuracy in double precision are:
97 !       factr=1.d+12 for low accuracy;
98 !             1.d+7  for moderate accuracy;
99 !             1.d+1  for extremely high accuracy.
100 !       The user can suppress this termination test by setting factr=0.
101 !
102 !     pgtol is a double precision variable.
103 !       On entry pgtol >= 0 is specified by the user.  The iteration
104 !         will stop when
105 !
106 !                 max{|proj g_i | i = 1, ..., n} <= pgtol
107 !
108 !         where pg_i is the ith component of the projected gradient.
109 !       The user can suppress this termination test by setting pgtol=0.
110 !
111 !     wa is a DOUBLE PRECISION  array of length
112 !       (2mmax + 5)nmax + 11mmax^2 + 8mmax used as workspace.
113 !       This array must not be altered by the user.
114 !
115 !     iwa is an INTEGER  array of length 3nmax used as
116 !       workspace. This array must not be altered by the user.
117 !
118 !     task is a CHARACTER string of length 60.
119 !       On first entry, it must be set to 'START'.
120 !       On a return with task(1:2)='FG', the user must evaluate the
121 !         function f and gradient g at the returned value of x.
122 !       On a return with task(1:5)='NEW_X', an iteration of the
123 !         algorithm has concluded, and f and g contain f(x) and g(x)
124 !         respectively.  The user can decide whether to continue or stop
125 !         the iteration.
126 !       When
127 !         task(1:4)='CONV', the termination test in L-BFGS-B has been
128 !           satisfied;
129 !         task(1:4)='ABNO', the routine has terminated abnormally
130 !           without being able to satisfy the termination conditions,
131 !           x contains the best approximation found,
132 !           f and g contain f(x) and g(x) respectively;
133 !         task(1:5)='ERROR', the routine has detected an error in the
134 !           input parameters;
135 !       On exit with task = 'CONV', 'ABNO' or 'ERROR', the variable task
136 !         contains additional information that the user can print.
137 !       This array should not be altered unless the user wants to
138 !          stop the run for some reason.  See driver2 or driver3
139 !          for a detailed explanation on how to stop the run
140 !          by assigning task(1:4)='STOP' in the driver.
141 !
142 !     iprint is an INTEGER variable that must be set by the user.
143 !       It controls the frequency and type of output generated:
```

```
144 !        iprint<0    no output is generated;
145 !        iprint=0    print only one line at the last iteration;
146 !        0<iprint<99 print also f and |proj g| every iprint iterations;
147 !        iprint=99   print details of every iteration except n-vectors;
148 !        iprint=100  print also the changes of active set and final x;
149 !        iprint>100  print details of every iteration including x and g;
150 !      When iprint > 0, the file iterate.dat will be created to
151 !                       summarize the iteration.
152 !
153 !     csave  is a CHARACTER working array of length 60.
154 !
155 !     lsave is a LOGICAL working array of dimension 4.
156 !       On exit with task = 'NEW_X', the following information is
157 !         available:
158 !       lsave(1) = .true.  the initial x did not satisfy the bounds;
159 !       lsave(2) = .true.  the problem contains bounds;
160 !       lsave(3) = .true.  each variable has upper and lower bounds.
161 !
162 !     isave is an INTEGER working array of dimension 44.
163 !       On exit with task = 'NEW_X', it contains information that
164 !       the user may want to access:
165 !         isave(30) = the current iteration number;
166 !         isave(34) = the total number of function and gradient
167 !                     evaluations;
168 !         isave(36) = the number of function value or gradient
169 !                                  evaluations in the current iteration;
170 !         isave(38) = the number of free variables in the current
171 !                     iteration;
172 !         isave(39) = the number of active constraints at the current
173 !                     iteration;
174 !
175 !         see the subroutine setulb.f for a description of other
176 !         information contained in isave
177 !
178 !     dsave is a DOUBLE PRECISION working array of dimension 29.
179 !       On exit with task = 'NEW_X', it contains information that
180 !         the user may want to access:
181 !         dsave(2)  = the value of f at the previous iteration;
182 !         dsave(5)  = the machine precision epsmch generated by the code;
183 !         dsave(13) = the infinity norm of the projected gradient;
184 !
185 !         see the subroutine setulb.f for a description of other
186 !         information contained in dsave
187 !
188 !     ------------------------------------------------------------
189 !           END OF THE DESCRIPTION OF THE VARIABLES IN L-BFGS-B
190 !     ------------------------------------------------------------
191 !
192       program driver
193 !
194 !     This simple driver demonstrates how to call the L-BFGS-B code to
195 !       solve a sample problem (the extended Rosenbrock function
196 !       subject to bounds on the variables). The dimension n of this
197 !       problem is variable.
198
199       implicit none
200 !
201 !     Declare variables and parameters needed by the code.
202 !       Note thar we wish to have output at every iteration.
203 !          iprint=1
204 !
205 !       We also specify the tolerances in the stopping criteria.
206 !          factr  = 1.0d+7, pgtol  = 1.0d-5
207 !
208 !       A description of all these variables is given at the beginning
209 !       of  the driver
210 !
211       integer,  parameter    :: n = 25, m = 5, iprint = 1
212       integer,  parameter    :: dp = kind(1.0d0)
213       real(dp), parameter    :: factr  = 1.0d+7, pgtol  = 1.0d-5
214 !
215       character(len=60)      :: task, csave
216       logical                :: lsave(4)
217       integer                :: isave(44)
218       real(dp)               :: f
219       real(dp)               :: dsave(29)
220       integer,  allocatable  :: nbd(:), iwa(:)
221       real(dp), allocatable  :: x(:), l(:), u(:), g(:), wa(:)
222
223 !     Declare a few additional variables for this sample problem
224
225       real(dp)               :: t1, t2
226       integer                :: i
227
228 !     Allocate dynamic arrays
229
230       allocate ( nbd(n), x(n), l(n), u(n), g(n) )
```

```
231       allocate ( iwa(3*n) )
232       allocate ( wa(2*m*n + 11*m*m + 5*n + 8*m) )
233 !
234       do 10 i=1, n, 2
235          nbd(i) = 2
236          l(i)   = 1.0d0
237          u(i)   = 1.0d2
238    10 continue
239
240 !     Next set bounds on the even-numbered variables.
241
242       do 12 i=2, n, 2
243          nbd(i) =  2
244          l(i)   = -1.0d2
245          u(i)   =  1.0d2
246    12  continue
247
248 !     We now define the starting point.
249
250       do 14 i=1, n
251          x(i) = 3.0d0
252    14 continue
253
254       write (6,16)
255    16 format(/,5x, 'Solving sample problem.', &
256              /,5x, ' (f = 0.0 at the optimal solution.)',/)
257
258 !     We start the iteration by initializing task.
259
260       task = 'START'
261
262 !     The beginning of the loop
263
264       do while(task(1:2).eq.'FG'.or.task.eq.'NEW_X'.or. &
265                task.eq.'START')
266
267 !     This is the call to the L-BFGS-B code.
268
269          call setulb ( n, m, x, l, u, nbd, f, g, factr, pgtol, &
270                        wa, iwa, task, iprint,&
271                        csave, lsave, isave, dsave )
272
273          if (task(1:2) .eq. 'FG') then
274
275             f=.25d0*( x(1)-1.d0 )**2
276             do 20 i=2, n
277                f = f + ( x(i)-x(i-1 )**2 )**2
278    20       continue
279             f = 4.d0*f
280
281 !     Compute gradient g for the sample problem.
282
283             t1   = x(2) - x(1)**2
284             g(1) = 2.d0*(x(1) - 1.d0) - 1.6d1*x(1)*t1
285             do 22 i=2, n-1
286                t2   = t1
287                t1   = x(i+1) - x(i)**2
288                g(i) = 8.d0*t2 - 1.6d1*x(i)*t1
289    22       continue
290             g(n) = 8.d0*t1
291
292          end if
293       end do
294
295 !     end of loop do while
296
297
298       end program driver
299
```

4.3 driver2.f

This driver shows how to replace the default stopping test by other termination criteria. It also illustrates how to print the values of several parameters during the course of the iteration. The sample problem used here is the same as in DRIVER1 (the extended Rosenbrock function with bounds on the variables). (Fortran-77 version)

```
1 c> \file driver2.f
2
3 c
4 c  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 c  or "3-clause license")
6 c  Please read attached file License.txt
```

```
7 c
8 c                           DRIVER 2 in Fortran 77
9 c     ------------------------------------------------------------
10 c              CUSTOMIZED DRIVER FOR L-BFGS-B (version 3.0)
11 c     ------------------------------------------------------------
12 c
13 c        L-BFGS-B is a code for solving large nonlinear optimization
14 c             problems with simple bounds on the variables.
15 c
16 c        The code can also be used for unconstrained problems and is
17 c        as efficient for these problems as the earlier limited memory
18 c                         code L-BFGS.
19 c
20 c        This driver illustrates how to control the termination of the
21 c        run and how to design customized output.
22 c
23 c     References:
24 c
25 c        [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
26 c        memory algorithm for bound constrained optimization",
27 c        SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
28 c
29 c        [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
30 c        Subroutines for Large Scale Bound Constrained Optimization"
31 c        Tech. Report, NAM-11, EECS Department, Northwestern University,
32 c        1994.
33 c
34 c
35 c          (Postscript files of these papers are available via anonymous
36 c           ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
37 c
38 c                               *  *  *
39 c
40 c         February 2011   (latest revision)
41 c         Optimization Center at Northwestern University
42 c         Instituto Tecnologico Autonomo de Mexico
43 c
44 c         Jorge Nocedal and Jose Luis Morales
45 c         Jorge Nocedal and Jose Luis Morales, Remark on "Algorithm 778:
46 c         L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained
47 c         Optimization"  (2011). To appear in  ACM Transactions on
48 c         Mathematical Software,
49 c
50 c     **************
51
52       program driver
53
54 c     This driver shows how to replace the default stopping test
55 c       by other termination criteria. It also illustrates how to
56 c       print the values of several parameters during the course of
57 c       the iteration. The sample problem used here is the same as in
58 c       DRIVER1 (the extended Rosenbrock function with bounds on the
59 c       variables).
60
61       integer          nmax, mmax
62       parameter(nmax=1024, mmax=17)
63 c        nmax is the dimension of the largest problem to be solved.
64 c        mmax is the maximum number of limited memory corrections.
65
66 c     Declare the variables needed by the code.
67 c       A description of all these variables is given at the end of
68 c       driver1.
69
70       character*60     task, csave
71       logical          lsave(4)
72       integer          n, m, iprint,
73      +                 nbd(nmax), iwa(3*nmax), isave(44)
74       double precision f, factr, pgtol,
75      +                 x(nmax), l(nmax), u(nmax), g(nmax), dsave(29),
76      +                 wa(2*mmax*nmax+5*nmax+11*mmax*mmax+8*mmax)
77
78 c     Declare a few additional variables for the sample problem.
79
80       double precision t1, t2
81       integer          i
82
83 c     We suppress the default output.
84
85       iprint = -1
86
87 c     We suppress both code-supplied stopping tests because the
88 c        user is providing his own stopping criteria.
89
90       factr=0.0d0
91       pgtol=0.0d0
92
93 c     We specify the dimension n of the sample problem and the number
```

```
c        m of limited memory corrections stored.  (n and m should not
c        exceed the limits nmax and mmax respectively.)

      n=25
      m=5

c     We now specify nbd which defines the bounds on the variables:
c                    l   specifies the lower bounds,
c                    u   specifies the upper bounds.

c     First set bounds on the odd numbered variables.

      do 10 i=1,n,2
         nbd(i)=2
         l(i)=1.0d0
         u(i)=1.0d2
   10  continue

c     Next set bounds on the even numbered variables.

      do 12 i=2,n,2
         nbd(i)=2
         l(i)=-1.0d2
         u(i)=1.0d2
   12   continue

c     We now define the starting point.

      do 14 i=1,n
         x(i)=3.0d0
   14  continue

c     We now write the heading of the output.

      write (6,16)
   16  format(/,5x, 'Solving sample problem.',
     +       /,5x, ' (f = 0.0 at the optimal solution.)',/)

c     We start the iteration by initializing task.
c
      task = 'START'

c        ------- the beginning of the loop ----------

  111  continue

c     This is the call to the L-BFGS-B code.

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

      if (task(1:2) .eq. 'FG') then
c        the minimization routine has returned to request the
c        function f and gradient g values at the current x.

c        Compute function value f for the sample problem.

         f=.25d0*(x(1)-1.d0)**2
         do 20 i=2,n
            f=f+(x(i)-x(i-1)**2)**2
   20     continue
         f=4.d0*f

c        Compute gradient g for the sample problem.

         t1=x(2)-x(1)**2
         g(1)=2.d0*(x(1)-1.d0)-1.6d1*x(1)*t1
         do 22 i=2,n-1
            t2=t1
            t1=x(i+1)-x(i)**2
            g(i)=8.d0*t2-1.6d1*x(i)*t1
   22     continue
         g(n)=8.d0*t1

c          go back to the minimization routine.
         goto 111
      endif
c
      if (task(1:5) .eq. 'NEW_X') then
c
c        the minimization routine has returned with a new iterate.
c        At this point have the opportunity of stopping the iteration
c        or observing the values of certain parameters
c
c        First are two examples of stopping tests.

c        Note: task(1:4) must be assigned the value 'STOP' to terminate
```

```
181 c           the iteration and ensure that the final results are
182 c           printed in the default format. The rest of the character
183 c           string TASK may be used to store other information.
184
185 c         1) Terminate if the total number of f and g evaluations
186 c              exceeds 99.
187
188           if (isave(34) .ge. 99)
189      +      task='STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT'
190
191 c         2) Terminate if  |proj g|/(1+|f|) < 1.0d-10, where
192 c            "proj g" denoted the projected gradient
193
194           if (dsave(13) .le. 1.d-10*(1.0d0 + abs(f)))
195      +      task='STOP: THE PROJECTED GRADIENT IS SUFFICIENTLY SMALL'
196
197 c         We now wish to print the following information at each
198 c         iteration:
199 c
200 c           1) the current iteration number, isave(30),
201 c           2) the total number of f and g evaluations, isave(34),
202 c           3) the value of the objective function f,
203 c           4) the norm of the projected gradient,  dsve(13)
204 c
205 c         See the comments at the end of driver1 for a description
206 c         of the variables isave and dsave.
207
208           write (6,'(2(a,i5,4x),a,1p,d12.5,4x,a,1p,d12.5)') 'Iterate'
209      +      ,isave(30),'nfg =',isave(34),'f =',f,'|proj g| =',dsave(13)
210
211 c         If the run is to be terminated, we print also the information
212 c         contained in task as well as the final value of x.
213
214           if (task(1:4) .eq. 'STOP') then
215              write (6,*) task
216              write (6,*) 'Final X='
217              write (6,'((1x,1p, 6(1x,d11.4)))') (x(i),i = 1,n)
218           endif
219
220 c           go back to the minimization routine.
221           goto 111
222
223        endif
224
225 c            ---------- the end of the loop -------------
226
227 c      If task is neither FG nor NEW_X we terminate execution.
228
229        stop
230
231        end
232
233 c======================= The end of driver2 ============================
234
```

4.4 driver2.f90

This driver shows how to replace the default stopping test by other termination criteria. It also illustrates how to print the values of several parameters during the course of the iteration. The sample problem used here is the same as in DRIVER1 (the extended Rosenbrock function with bounds on the variables). (Fortran-90 version)

```
1 !> \file driver2.f90
2
3 !
4 !  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 !  or "3-clause license")
6 !  Please read attached file License.txt
7 !
8 !
9 !                  DRIVER 2  in Fortran 90
10 !    ------------------------------------------------------------
11 !             CUSTOMIZED DRIVER FOR L-BFGS-B
12 !    ------------------------------------------------------------
13 !
14 !       L-BFGS-B is a code for solving large nonlinear optimization
15 !            problems with simple bounds on the variables.
16 !
17 !       The code can also be used for unconstrained problems and is
18 !       as efficient for these problems as the earlier limited memory
19 !                         code L-BFGS.
20 !
21 !       This driver illustrates how to control the termination of the
```

```
!       run and how to design customized output.
!
!    References:
!
!       [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
!       memory algorithm for bound constrained optimization",
!       SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
!
!       [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
!       Subroutines for Large Scale Bound Constrained Optimization"
!       Tech. Report, NAM-11, EECS Department, Northwestern University,
!       1994.
!
!
!         (Postscript files of these papers are available via anonymous
!          ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
!
!                              *  *  *
!
!         February 2011   (latest revision)
!         Optimization Center at Northwestern University
!         Instituto Tecnologico Autonomo de Mexico
!
!         Jorge Nocedal and Jose Luis Morales
!
!    **************
      program driver

!     This driver shows how to replace the default stopping test
!     by other termination criteria. It also illustrates how to
!     print the values of several parameters during the course of
!     the iteration. The sample problem used here is the same as in
!     DRIVER1 (the extended Rosenbrock function with bounds on the
!     variables).

      implicit none

!     Declare variables and parameters needed by the code.
!
!     Note that we suppress the default output (iprint = -1)
!     We suppress both code-supplied stopping tests because the
!     user is providing his/her own stopping criteria.

      integer,  parameter    :: n = 25, m = 5, iprint = -1
      integer,  parameter    :: dp = kind(1.0d0)
      real(dp), parameter    :: factr  = 0.0d0, pgtol  = 0.0d0

      character(len=60)      :: task, csave
      logical                :: lsave(4)
      integer                :: isave(44)
      real(dp)               :: f
      real(dp)               :: dsave(29)
      integer,  allocatable  :: nbd(:), iwa(:)
      real(dp), allocatable  :: x(:), l(:), u(:), g(:), wa(:)
!
      real(dp)               :: t1, t2
      integer                :: i

      allocate ( nbd(n), x(n), l(n), u(n), g(n) )
      allocate ( iwa(3*n) )
      allocate ( wa(2*m*n + 5*n + 11*m*m + 8*m) )
!
!    This driver shows how to replace the default stopping test
!    by other termination criteria. It also illustrates how to
!    print the values of several parameters during the course of
!    the iteration. The sample problem used here is the same as in
!    DRIVER1 (the extended Rosenbrock function with bounds on the
!    variables).
!    We now specify nbd which defines the bounds on the variables:
!                    l   specifies the lower bounds,
!                    u   specifies the upper bounds.
!
!    First set bounds on the odd numbered variables.

       do 10 i=1, n,2
          nbd(i)=2
          l(i)=1.0d0
          u(i)=1.0d2
   10  continue
!
!    Next set bounds on the even numbered variables.

       do 12 i=2, n,2
          nbd(i)=2
          l(i)=-1.0d2
          u(i)=1.0d2
   12   continue
```

```
109
110 !     We now define the starting point.
111
112       do 14 i=1, n
113          x(i)=3.0d0
114    14 continue
115
116 !     We now write the heading of the output.
117
118       write (6,16)
119    16 format(/,5x, 'Solving sample problem.', &
120              /,5x, ' (f = 0.0 at the optimal solution.)',/)
121
122
123 !     We start the iteration by initializing task.
124 !
125       task = 'START'
126
127 !        ------- the beginning of the loop ----------
128
129       do while( task(1:2).eq.'FG'.or.task.eq.'NEW_X'.or. &
130                 task.eq.'START')
131
132 !     This is the call to the L-BFGS-B code.
133
134          call setulb(n,m,x,l,u,nbd,f,g,factr,pgtol,wa,iwa,task,iprint, &
135                      csave,lsave,isave,dsave)
136
137          if (task(1:2) .eq. 'FG') then
138
139 !        the minimization routine has returned to request the
140 !        function f and gradient g values at the current x.
141
142 !        Compute function value f for the sample problem.
143
144             f =.25d0*(x(1) - 1.d0)**2
145             do 20 i=2,n
146                f = f + (x(i) - x(i-1)**2)**2
147    20       continue
148             f = 4.d0*f
149
150 !        Compute gradient g for the sample problem.
151
152             t1 = x(2) - x(1)**2
153             g(1) = 2.d0*(x(1) - 1.d0) - 1.6d1*x(1)*t1
154             do 22 i= 2,n-1
155                t2 = t1
156                t1 = x(i+1) - x(i)**2
157                g(i) = 8.d0*t2 - 1.6d1*x(i)*t1
158    22       continue
159             g(n)=8.d0*t1
160 !
161       else
162 !
163          if (task(1:5) .eq. 'NEW_X') then
164 !
165 !       the minimization routine has returned with a new iterate.
166 !       At this point have the opportunity of stopping the iteration
167 !       or observing the values of certain parameters
168 !
169 !       First are two examples of stopping tests.
170
171 !       Note: task(1:4) must be assigned the value 'STOP' to terminate
172 !         the iteration and ensure that the final results are
173 !         printed in the default format. The rest of the character
174 !         string TASK may be used to store other information.
175
176 !       1) Terminate if the total number of f and g evaluations
177 !            exceeds 99.
178
179             if (isave(34) .ge. 99)  &
180                task='STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT'
181
182 !       2) Terminate if  |proj g|/(1+|f|) < 1.0d-10, where
183 !          "proj g" denoted the projected gradient
184
185             if (dsave(13) .le. 1.d-10*(1.0d0 + abs(f))) &
186               task='STOP: THE PROJECTED GRADIENT IS SUFFICIENTLY SMALL'
187
188 !       We now wish to print the following information at each
189 !       iteration:
190 !
191 !         1) the current iteration number, isave(30),
192 !         2) the total number of f and g evaluations, isave(34),
193 !         3) the value of the objective function f,
194 !         4) the norm of the projected gradient,  dsve(13)
195 !
```

```
196 !       See the comments at the end of driver1 for a description
197 !       of the variables isave and dsave.
198
199             write (6,'(2(a,i5,4x),a,1p,d12.5,4x,a,1p,d12.5)') 'Iterate' &
200                , isave(30),'nfg =',isave(34),'f =',f,'|proj g| =',dsave(13)
201
202 !        If the run is to be terminated, we print also the information
203 !        contained in task as well as the final value of x.
204
205             if (task(1:4) .eq. 'STOP') then
206                write (6,*) task
207                write (6,*) 'Final X='
208                write (6,'((1x,1p, 6(1x,d11.4)))') (x(i),i = 1,n)
209             end if
210
211          end if
212       end if
213
214       end do
215 !           ---------- the end of the loop -------------
216
217 !     If task is neither FG nor NEW_X we terminate execution.
218
219       end program driver
220
221 !======================= The end of driver2 ============================
222
```

4.5 driver3.f

This time-controlled driver shows that it is possible to terminate a run by elapsed CPU time, and yet be able to print all desired information. This driver also illustrates the use of two stopping criteria that may be used in conjunction with a limit on execution time. The sample problem used here is the same as in driver1 and driver2 (the extended Rosenbrock function with bounds on the variables). (Fortran-77 version)

```
1 c> \file driver3.f
2
3 c
4 c  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 c  or "3-clause license")
6 c  Please read attached file License.txt
7 c
8 c                            DRIVER 3 in Fortran 77
9 c     ------------------------------------------------------------
10 c            TIME-CONTROLLED DRIVER FOR L-BFGS-B (version 3.0)
11 c     ------------------------------------------------------------
12 c
13 c        L-BFGS-B is a code for solving large nonlinear optimization
14 c             problems with simple bounds on the variables.
15 c
16 c        The code can also be used for unconstrained problems and is
17 c        as efficient for these problems as the earlier limited memory
18 c                          code L-BFGS.
19 c
20 c        This driver shows how to terminate a run after some prescribed
21 c        CPU time has elapsed, and how to print the desired information
22 c        before exiting.
23 c
24 c     References:
25 c
26 c        [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
27 c        memory algorithm for bound constrained optimization",
28 c        SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
29 c
30 c        [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
31 c        Subroutines for Large Scale Bound Constrained Optimization"
32 c        Tech. Report, NAM-11, EECS Department, Northwestern University,
33 c        1994.
34 c
35 c
36 c          (Postscript files of these papers are available via anonymous
37 c           ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
38 c
39 c                               *  *  *
40 c
41 c         February 2011   (latest revision)
42 c         Optimization Center at Northwestern University
43 c         Instituto Tecnologico Autonomo de Mexico
44 c
45 c         Jorge Nocedal and Jose Luis Morales, Remark on "Algorithm 778:
46 c         L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained
47 c         Optimization"  (2011). To appear in  ACM Transactions on
```

```
c         Mathematical Software,
c
c
c     **************

      program driver

c     This time-controlled driver shows that it is possible to terminate
c     a run by elapsed CPU time, and yet be able to print all desired
c     information. This driver also illustrates the use of two
c     stopping criteria that may be used in conjunction with a limit
c     on execution time. The sample problem used here is the same as in
c     driver1 and driver2 (the extended Rosenbrock function with bounds
c     on the variables).

      integer          nmax, mmax
      parameter(nmax=1024,mmax=17)
c        nmax is the dimension of the largest problem to be solved.
c        mmax is the maximum number of limited memory corrections.

c     Declare the variables needed by the code.
c       A description of all these variables is given at the end of
c       driver1.

      character*60     task, csave
      logical          lsave(4)
      integer          n, m, iprint,
     +                 nbd(nmax), iwa(3*nmax), isave(44)
      double precision f, factr, pgtol,
     +                 x(nmax), l(nmax), u(nmax), g(nmax), dsave(29),
     +                 wa(2*mmax*nmax+5*nmax+11*mmax*mmax+8*mmax)

c     Declare a few additional variables for the sample problem
c       and for keeping track of time.

      double precision t1, t2, time1, time2, tlimit
      integer          i, j

c     We specify a limite on the CPU time (in seconds).

      tlimit = 0.2

c     We suppress the default output.  (The user could also elect to
c       use the default output by choosing iprint >= 0.)

      iprint = -1

c     We suppress the code-supplied stopping tests because we will
c       provide our own termination conditions

      factr=0.0d0
      pgtol=0.0d0

c     We specify the dimension n of the sample problem and the number
c        m of limited memory corrections stored.  (n and m should not
c        exceed the limits nmax and mmax respectively.)

      n=1000
      m=10

c     We now specify nbd which defines the bounds on the variables:
c                    l   specifies the lower bounds,
c                    u   specifies the upper bounds.

c     First set bounds on the odd-numbered variables.

      do 10 i=1,n,2
         nbd(i)=2
         l(i)=1.0d0
         u(i)=1.0d2
   10  continue

c     Next set bounds on the even-numbered variables.

      do 12 i=2,n,2
         nbd(i)=2
         l(i)=-1.0d2
         u(i)=1.0d2
   12   continue

c     We now define the starting point.

      do 14 i=1,n
         x(i)=3.0d0
   14  continue

c     We now write the heading of the output.
```

```
135
136       write (6,16)
137    16  format(/,5x, 'Solving sample problem.',
138      +       /,5x, ' (f = 0.0 at the optimal solution.)',/)
139
140 c     We start the iteration by initializing task.
141 c
142       task = 'START'
143
144 c        ------- the beginning of the loop ----------
145
146 c     We begin counting the CPU time.
147
148       call timer(time1)
149
150  111  continue
151
152 c     This is the call to the L-BFGS-B code.
153
154       call setulb(n,m,x,l,u,nbd,f,g,factr,pgtol,wa,iwa,task,iprint,
155      +            csave,lsave,isave,dsave)
156
157       if (task(1:2) .eq. 'FG') then
158 c        the minimization routine has returned to request the
159 c        function f and gradient g values at the current x.
160 c        Before evaluating f and g we check the CPU time spent.
161
162          call timer(time2)
163          if (time2-time1 .gt. tlimit) then
164             task='STOP: CPU EXCEEDING THE TIME LIMIT.'
165
166 c          Note: Assigning task(1:4)='STOP' will terminate the run;
167 c          setting task(7:9)='CPU' will restore the information at
168 c          the latest iterate generated by the code so that it can
169 c          be correctly printed by the driver.
170
171 c          In this driver we have chosen to disable the
172 c          printing options of the code (we set iprint=-1);
173 c          instead we are using customized output: we print the
174 c          latest value of x, the corresponding function value f and
175 c          the norm of the projected gradient |proj g|.
176
177 c          We print out the information contained in task.
178
179             write (6,*) task
180
181 c          We print the latest iterate contained in wa(j+1:j+n), where
182 c
183             j = 3*n+2*m*n+11*m**2
184             write (6,*) 'Latest iterate X ='
185             write (6,'((1x,1p, 6(1x,d11.4)))') (wa(i),i = j+1,j+n)
186
187 c          We print the function value f and the norm of the projected
188 c          gradient |proj g| at the last iterate; they are stored in
189 c          dsave(2) and dsave(13) respectively.
190
191             write (6,'(a,1p,d12.5,4x,a,1p,d12.5)')
192      +         'At latest iterate   f =',dsave(2),'|proj g| =',dsave(13)
193
194          else
195
196 c          The time limit has not been reached and we compute
197 c          the function value f for the sample problem.
198
199             f=.25d0*(x(1)-1.d0)**2
200             do 20 i=2,n
201                f=f+(x(i)-x(i-1)**2)**2
202    20       continue
203             f=4.d0*f
204
205 c          Compute gradient g for the sample problem.
206
207             t1=x(2)-x(1)**2
208             g(1)=2.d0*(x(1)-1.d0)-1.6d1*x(1)*t1
209             do 22 i=2,n-1
210                t2=t1
211                t1=x(i+1)-x(i)**2
212                g(i)=8.d0*t2-1.6d1*x(i)*t1
213    22       continue
214             g(n)=8.d0*t1
215
216          endif
217
218 c          go back to the minimization routine.
219          goto 111
220       endif
221 c
```

```
222       if (task(1:5) .eq. 'NEW_X') then
223 c        the minimization routine has returned with a new iterate.
224 c        The time limit has not been reached, and we test whether
225 c        the following two stopping tests are satisfied:
226
227 c        1) Terminate if the total number of f and g evaluations
228 c             exceeds 900.
229
230          if (isave(34) .ge. 900)
231      +      task='STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT'
232
233 c        2) Terminate if  |proj g|/(1+|f|) < 1.0d-10.
234
235          if (dsave(13) .le. 1.d-10*(1.0d0 + abs(f)))
236      +      task='STOP: THE PROJECTED GRADIENT IS SUFFICIENTLY SMALL'
237
238 c        We wish to print the following information at each iteration:
239 c          1) the current iteration number, isave(30),
240 c          2) the total number of f and g evaluations, isave(34),
241 c          3) the value of the objective function f,
242 c          4) the norm of the projected gradient,  dsve(13)
243 c
244 c        See the comments at the end of driver1 for a description
245 c        of the variables isave and dsave.
246
247
248          write (6,'(2(a,i5,4x),a,1p,d12.5,4x,a,1p,d12.5)') 'Iterate'
249      +      ,isave(30),'nfg =',isave(34),'f =',f,'|proj g| =',dsave(13)
250
251 c        If the run is to be terminated, we print also the information
252 c        contained in task as well as the final value of x.
253
254
255          if (task(1:4) .eq. 'STOP') then
256             write (6,*) task
257             write (6,*) 'Final X='
258             write (6,'((1x,1p, 6(1x,d11.4)))') (x(i),i = 1,n)
259          endif
260
261 c          go back to the minimization routine.
262          goto 111
263
264       endif
265
266 c           ---------- the end of the loop -------------
267
268 c     If task is neither FG nor NEW_X we terminate execution.
269
270       stop
271
272       end
273
274 c======================= The end of driver3 ============================
275
```

4.6 driver3.f90

This time-controlled driver shows that it is possible to terminate a run by elapsed CPU time, and yet be able to print all desired information. This driver also illustrates the use of two stopping criteria that may be used in conjunction with a limit on execution time. The sample problem used here is the same as in driver1 and driver2 (the extended Rosenbrock function with bounds on the variables). (Fortran-90 version)

```
1 !> \file driver3.f90
2
3 !
4 !  L-BFGS-B is released under the "New BSD License" (aka "Modified BSD License"
5 !  or "3-clause license")
6 !  Please read attached file License.txt
7 !
8 !                            DRIVER 3  in Fortran 90
9 !     ----------------------------------------------------------------
10 !           TIME-CONTROLLED DRIVER FOR L-BFGS-B
11 !     ----------------------------------------------------------------
12 !
13 !        L-BFGS-B is a code for solving large nonlinear optimization
14 !             problems with simple bounds on the variables.
15 !
16 !        The code can also be used for unconstrained problems and is
17 !        as efficient for these problems as the earlier limited memory
18 !                         code L-BFGS.
19 !
20 !        This driver shows how to terminate a run after some prescribed
```

```
21 !        CPU time has elapsed, and how to print the desired information
22 !        before exiting.
23 !
24 !     References:
25 !
26 !        [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited
27 !        memory algorithm for bound constrained optimization",
28 !        SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.
29 !
30 !        [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, "L-BFGS-B: FORTRAN
31 !        Subroutines for Large Scale Bound Constrained Optimization"
32 !        Tech. Report, NAM-11, EECS Department, Northwestern University,
33 !        1994.
34 !
35 !
36 !          (Postscript files of these papers are available via anonymous
37 !           ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)
38 !
39 !                               *  *  *
40 !
41 !         February 2011   (latest revision)
42 !         Optimization Center at Northwestern University
43 !         Instituto Tecnologico Autonomo de Mexico
44 !
45 !         Jorge Nocedal and Jose Luis Morales
46 !
47 !     **************
48
49       program driver
50
51 !     This time-controlled driver shows that it is possible to terminate
52 !     a run by elapsed CPU time, and yet be able to print all desired
53 !     information. This driver also illustrates the use of two
54 !     stopping criteria that may be used in conjunction with a limit
55 !     on execution time. The sample problem used here is the same as in
56 !     driver1 and driver2 (the extended Rosenbrock function with bounds
57 !     on the variables).
58
59       implicit none
60
61 !     We specify a limit on the CPU time (tlimit = 10 seconds)
62 !
63 !     We suppress the default output (iprint = -1). The user could
64 !       also elect to use the default output by choosing iprint >= 0.)
65 !     We suppress the code-supplied stopping tests because we will
66 !       provide our own termination conditions
67 !     We specify the dimension n of the sample problem and the number
68 !        m of limited memory corrections stored.
69
70       integer,  parameter    :: n = 1000, m = 10, iprint = -1
71       integer,  parameter    :: dp = kind(1.0d0)
72       real(dp), parameter    :: factr  = 0.0d0, pgtol  = 0.0d0, &
73                                 tlimit = 10.0d0
74 !
75       character(len=60)      :: task, csave
76       logical                :: lsave(4)
77       integer                :: isave(44)
78       real(dp)               :: f
79       real(dp)               :: dsave(29)
80       integer,  allocatable  :: nbd(:), iwa(:)
81       real(dp), allocatable  :: x(:), l(:), u(:), g(:), wa(:)
82 !
83       real(dp)               :: t1, t2, time1, time2
84       integer                :: i, j
85
86       allocate ( nbd(n), x(n), l(n), u(n), g(n) )
87       allocate ( iwa(3*n) )
88       allocate ( wa(2*m*n + 5*n + 11*m*m + 8*m) )
89
90 !     This time-controlled driver shows that it is possible to terminate
91 !     a run by elapsed CPU time, and yet be able to print all desired
92 !     information. This driver also illustrates the use of two
93 !     stopping criteria that may be used in conjunction with a limit
94 !     on execution time. The sample problem used here is the same as in
95 !     driver1 and driver2 (the extended Rosenbrock function with bounds
96 !     on the variables).
97
98 !     We now specify nbd which defines the bounds on the variables:
99 !                     l   specifies the lower bounds,
100 !                     u   specifies the upper bounds.
101
102 !     First set bounds on the odd-numbered variables.
103
104       do 10 i=1, n,2
105          nbd(i)=2
106          l(i)=1.0d0
107          u(i)=1.0d2
```

```
  10  continue

!     Next set bounds on the even-numbered variables.

      do 12 i=2, n,2
         nbd(i)=2
         l(i)=-1.0d2
         u(i)=1.0d2
  12   continue

!     We now define the starting point.

      do 14 i=1, n
         x(i)=3.0d0
  14  continue

!     We now write the heading of the output.

      write (6,16)
  16  format(/,5x, 'Solving sample problem.',&
             /,5x, ' (f = 0.0 at the optimal solution.)',/)

!     We start the iteration by initializing task.

      task = 'START'

!        ------- the beginning of the loop ----------

!     We begin counting the CPU time.

      call timer(time1)

      do while( task(1:2).eq.'FG'.or.task.eq.'NEW_X'.or. &
                task.eq.'START')

!     This is the call to the L-BFGS-B code.

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

         if (task(1:2) .eq. 'FG') then

!        the minimization routine has returned to request the
!        function f and gradient g values at the current x.
!        Before evaluating f and g we check the CPU time spent.

         call timer(time2)
         if (time2-time1 .gt. tlimit) then
            task='STOP: CPU EXCEEDING THE TIME LIMIT.'

!           Note: Assigning task(1:4)='STOP' will terminate the run;
!           setting task(7:9)='CPU' will restore the information at
!           the latest iterate generated by the code so that it can
!           be correctly printed by the driver.

!           In this driver we have chosen to disable the
!           printing options of the code (we set iprint=-1);
!           instead we are using customized output: we print the
!           latest value of x, the corresponding function value f and
!           the norm of the projected gradient |proj g|.

!           We print out the information contained in task.

               write (6,*) task

!           We print the latest iterate contained in wa(j+1:j+n), where

               j = 3*n+2*m*n+11*m**2
               write (6,*) 'Latest iterate X ='
               write (6,'((1x,1p, 6(1x,d11.4)))') (wa(i),i = j+1,j+n)

!           We print the function value f and the norm of the projected
!           gradient |proj g| at the last iterate; they are stored in
!           dsave(2) and dsave(13) respectively.

               write (6,'(a,1p,d12.5,4x,a,1p,d12.5)') &
               'At latest iterate   f =',dsave(2),'|proj g| =',dsave(13)
            else

!           The time limit has not been reached and we compute
!           the function value f for the sample problem.

               f=.25d0*(x(1)-1.d0)**2
               do 20 i=2, n
                  f=f+(x(i)-x(i-1)**2)**2
  20           continue
               f=4.d0*f
```

```
195
196 !          Compute gradient g for the sample problem.
197
198                t1 = x(2) - x(1)**2
199                g(1) = 2.d0*(x(1)-1.d0)-1.6d1*x(1)*t1
200                do 22 i=2,n-1
201                   t2=t1
202                   t1=x(i+1)-x(i)**2
203                   g(i)=8.d0*t2-1.6d1*x(i)*t1
204   22           continue
205                g(n)=8.d0*t1
206             endif
207
208 !          go back to the minimization routine.
209          else
210
211            if (task(1:5) .eq. 'NEW_X') then
212
213 !        the minimization routine has returned with a new iterate.
214 !        The time limit has not been reached, and we test whether
215 !        the following two stopping tests are satisfied:
216
217 !        1) Terminate if the total number of f and g evaluations
218 !             exceeds 900.
219
220              if (isave(34) .ge. 900) &
221              task='STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT'
222
223 !        2) Terminate if  |proj g|/(1+|f|) < 1.0d-10.
224
225              if (dsave(13) .le. 1.d-10*(1.0d0 + abs(f))) &
226              task='STOP: THE PROJECTED GRADIENT IS SUFFICIENTLY SMALL'
227
228 !        We wish to print the following information at each iteration:
229 !          1) the current iteration number, isave(30),
230 !          2) the total number of f and g evaluations, isave(34),
231 !          3) the value of the objective function f,
232 !          4) the norm of the projected gradient,  dsve(13)
233 !
234 !        See the comments at the end of driver1 for a description
235 !        of the variables isave and dsave.
236
237              write (6,'(2(a,i5,4x),a,1p,d12.5,4x,a,1p,d12.5)') 'Iterate' &
238              ,isave(30),'nfg =',isave(34),'f =',f,'|proj g| =',dsave(13)
239
240 !        If the run is to be terminated, we print also the information
241 !        contained in task as well as the final value of x.
242
243              if (task(1:4) .eq. 'STOP') then
244                 write (6,*) task
245                 write (6,*) 'Final X='
246                 write (6,'((1x,1p, 6(1x,d11.4)))') (x(i),i = 1,n)
247              endif
248
249           endif
250         end if
251       end do
252
253 !     If task is neither FG nor NEW_X we terminate execution.
254
255       end program driver
256
257 !======================= The end of driver3 ============================
258
```

Index