LANCELOT_simple,

A SIMPLE INTERFACE FOR LANCELOT B

by N. I. M. Gould[1], D. Orban[2] and Ph. L. Toint[3]

Report 07/12 5 December 2007

[1] Oxford University Computing Laboratory,
Wolfson Building, Parks Road,
Oxford OX1 3QD, England.
Email: nick.gould@comlab.ox.ac.uk

GERAD and
[2] Mathematics and Industrial Engineering Department,
Ecole Polytechnique de Montréal,
Montréal, Canada.
Email: dominique.orban@gerad.ca

[3] Department of Mathematics,
FUNDP-University of Namur,
61, rue de Bruxelles, B-5000 Namur, Belgium.
Email: philippe.toint@fundp.ac.be

LANCELOT_simple: a simple interface for LANCELOT B

N. I. M. Gould D. Orban Ph. L. Toint

5 December 2007

Abstract

We describe LANCELOT_simple, an interface to the LANCELOT B nonlinear optimization package within the GALAHAD library (Gould, Orban and Toint, 2003) which ignores problem structure. The result is an easy-to-use Fortran 90 subroutine, with a small number of intuitively interpretable arguments. However, since structure is ignored, the means of presenting problems to the solver limited and the choice of algorithmic parameters considerably restricted, the performance is likely to be sub-optimal. The proposed interface may nevertheless be useful for users interested in the solution of simple low-dimensional problems when ease of interface matters more than actual numerical efficiency.

1 Introduction

This note presents LANCELOT_simple, a simple and somewhat *naive* Fortran 90 interface to the sophisticated optimization package LANCELOT B, whose aim is to solve the nonlinear optimization problem

$$\min_{x\in\mathbb{R}^n} f(x),$$

possibly subject to constraints of the one or more of the forms

$$b_l \leq x \leq b_u,$$

$$c_e(x) = 0,$$

$$c_i(x) \leq 0,$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $c_e : \mathbb{R}^n \to \mathbb{R}^m$ and $c_i : \mathbb{R}^n \to \mathbb{R}^q$ are twice-continuously differentiable functions, and b_l, b_u are vectors of $\mathbb{R}^n$ whose components are allowed to be arbitrarily large in absolute value. LANCELOT B is one of the packages contained in the GALAHAD software library for numerical nonlinear optimization[(1)].

Why do we call this interface *naive*? At variance with more elaborate interfaces for LANCELOT, the one presented here completely *ignores any underlying partial separability or sparsity structure, limits the forms under which the problem can be presented to the solver* and drastically *restricts the range of available algorithmic options*. While undoubtedly simpler to use than its more elaborate counterpart, it may thus offer inferior numerical performance, especially for difficult/large problems, where structure exploitation and/or careful selection of algorithmic variants matter. Thus, be warned that

THE BEST PERFORMANCE OBTAINABLE WITH LANCELOT B IS MOST LIKELY NOT WITH THIS NAIVE INTERFACE.

[(1)] See http://galahad.rl.ac.uk/galahad-www/index.html.

2 How to use the interface

Before using LANCELOT_simple, its module must be made available:

```
USE LANCELOT_simple_double
```

The complete calling sequence[(2)] for LANCELOT_simple is as follows:

```
CALL LANCELOT_simple( n, X, fx, exit_code [, MY_FUN] [,MY_GRAD]      &
                     [,MY_HESS] [,BL] [,BU] [,VNAMES] [,CNAMES]      &
                     [,neq] [,nin] [,CX] [,Y] [,iters] [,maxit]      &
                     [,gradtol] [,feastol] [,print_level]            )
```

where the arguments within brackets are optional. The following sections indicate how this calling sequence can be used in commonly occuring contexts.

2.1 Unconstrained minimization problems

The user should provide, at the very least, suitable values for the following input arguments:

`n` (integer): the number n of variables,

`X` (double precision vector of size `n`): the starting point for the minimization

In addition the user must provide a subroutine to compute the objective function value at any given x, with the default name and interface

```
FUN( X, fx )
```

where `X(1:n)` contains the values of the variables x on input, and `fx` is a double precision scalar returning the value $f(x)$. If no further information is given, the problem is taken to be unconstrained, and derivatives of f are unavailable (gradients will be estimated by forward finite-differences by the LANCELOT package, and Hessian approximated using the Symmetric-Rank-One quasi-Newton update).

The best value of x found by LANCELOT B is returned to the user in the vector `X` and the associated objective function value in the double precision output argument `fx`. In addition, the integer output argument `exit_code` contains the exit status of the LANCELOT run, the value 0 indicating a successful run. Other values indicate errors in the input or unsuccessful runs, and are detailed in the specification documentation for LANCELOT B (with the exception of the value 19 in the constrained case discussed later, which reports a negative value for one or both input arguments `nin` and `neq`).

It is also possible to specify a different name for the objective function evaluation subroutine, replacing `FUN` by, say, `FUNPROB`. This is accomplished by specifying the optional argument `MY_FUN` to have the value `FUNPROB`. This is, in turn, best done by including the explicit declaration `MY_FUN = FUNPROB` in the calling sequence to LANCELOT_simple after the leading four compulsory arguments, as well as the declaring `FUNPROB` as external. Note that `FUNPROB` must have the same calling sequence as `FUN`.

If the user is able to provide the gradient of f, then the optional input argument `MY_GRAD` must be specified and given the name of the user-supplied routine for computing the gradient, say `GRADPROB`. This subroutine must have an interface of the form

```
GRADPROB( X, G )
```

where `G` is a double precision vector of size `n` in which the subroutine returns the value of the gradient of f at $x =$ `X`. The calling sequence to LANCELOT_simple must thus contain (in this case), `MY_GRAD = GRADPROB`, and a declaration of `GRADPROB` as external.

If, additionally, the user is willing to compute the second-derivative matrix of f at x, the optional input argument `MY_HESS` must be specified and given the name of the user-supplied routine computing the Hessian, say `HESSPROB`. This must have an interface of the form

[(2)] The reference package specification document is available in the GALAHAD installation tree in the file ./doc/lancelot_simple.pdf.

```
HESSPROB( X, H )
```

where `H` is a double precision vector of size `n*(n+1)/2` in which the subroutine returns the entries of the upper triangular part of the Hessian of f at x =`X`, stored by columns. The calling sequence to LANCELOT_simple must thus contain (in this case), `MY_HESS = HESSPROB`, and a declaration of `HESSPROB` as external. Note that first derivatives must also be available in this case, and `MY_GRAD` must thus be specified.

The names of the problem variables may be specified in the optional input argument

`VNAMES` (a size `n` vector of character fields of length 10)

by inserting `VNAMES = mynames` in the calling sequence.

One final feature is that by specifying the optional integer argument `iters = lanit`, the number of iterations performed by LANCELOT is recorded in `lanit`.

Example

Let us consider the optimization problem

$$\min_{x_1,x_2} f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

which is the renowned Rosenbrock "banana" problem. The most basic way to solve the problem (but *not* the most efficient) is, given the starting point `X = (/ -1.2d0, 1.0d0 /)`, to

```
CALL LANCELOT_simple( 2, X, fx, exit_code )
```

where the function `FUN` is given by

```
SUBROUTINE FUN( X, F )
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
    REAL( KIND = wp ), INTENT( IN )  :: X( : )
    REAL( KIND = wp ), INTENT( OUT ) :: F
    F = 100.0_wp*(X(2)-X(1)**2)**2 +(1.0_wp-X(1))**2
    RETURN
END SUBROUTINE FUN
```

After compiling and linking (with the GALAHAD modules and library), the solution is returned in 60 iterations (with `exit_code` = 0). The same effect is obtained by the call

```
CALL LANCELOT_simple( 2, X, fx, exit_code, MY_FUN = FUN )
```

but this syntax allows the change of name from `FUN` to, say, `ROSENBROCK_FUN` in the user-supplied program, as discussed above.

If we now wish to use first and second derivatives of the objective function, one could use the call

```
CALL LANCELOT_simple( 2, X, fx, exit_code, MY_FUN = FUN, &
                      MY_GRAD = ROSENBROCK_GRAD,         &
                      MY_HESS = ROSENBROCK_HESS          )
```

provide the additional routines

```
SUBROUTINE ROSENBROCK_GRAD( X, G )
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
    REAL( KIND = wp ), INTENT( IN )  :: X( : )
    REAL( KIND = wp ), INTENT( OUT ) :: G( : )
    G( 1 ) = -400.0_wp*(X(2)-X(1)**2)*X(1)-2.0_wp*(1.0_wp-X(1))
    G( 2 ) =  200.0_wp*(X(2)-X(1)**2)
    RETURN
END SUBROUTINE ROSENBROCK_GRAD
```

and

```
    SUBROUTINE ROSENBROCK_HESS( X, H )
        INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
        REAL( KIND = wp ), INTENT( IN )  :: X( : )
        REAL( KIND = wp ), INTENT( OUT ) :: H( : )
        H( 1 ) = -400.0_wp*(X(2)-3.0_wp*X(1)**2)+2.0_wp
        H( 2 ) = -400.0_wp*X(1)
        H( 3 ) =  200.0_wp
        RETURN
    END SUBROUTINE ROSENBROCK_HESS
```

and declare them external in the calling program.

Convergence is then obtained in 23 iterations. Note that using exact first-derivatives only is also possible: `MY_HESS` should then be absent from the calling sequence and providing the subroutine `ROSENBROCK_HESS` is unnecessary.

2.2 Bound constrained minimization problems

Bounds on the problem variables may be imposed by specifying one or both of

`BL` **(double precision vector of size** `n`**):** the lower bounds b_l on the variables,

`BU` **(double precision vector of size** `n`**):** the upper bounds b_u on the variables.

Note that infinite bounds (represented by a number larger than 10^{20} in absolute value) are acceptable, as well as equal lower and upper bounds, which amounts to fixing the corresponding variables. Except for the specification of `BL` and/or `BU`, the interface is identical to that for unconstrained problems described in the previous paragraph.

Example

If one now wishes to impose zero upper bounds on the variables of our unconstrained problem and give names to the variables and the objective function, one could use the following call

```
    VNAMES(1) = 'x1'
    VNAMES(2) = 'x2'
    CALL LANCELOT_simple( 2, X, fx, exit_code, MY_FUN = FUN, &
                          MY_GRAD = ROSENBROCK_GRAD,         &
                          MY_HESS = ROSENBROCK_HESS,         &
                          BU = (/ 0.0d0, 0.0d0 /),           &
                          VNAMES = VNAMES                    )
```

in which case convergence is obtained in 6 iterations.

2.3 Equality constrained minimization problems

If, additionally, general equality constraints are also present in the problem, this must be declared by specifying the following additional optional input argument

`neq` **(integer):** the number m of equality constraints.

In this case, the equality constraints are numbered from 1 to `neq` and the value of the i-th equality constraint must be computed by a user-supplied routine of the form

```
    FUNPROB( X, fx, i )                        ( i = 1,...,neq )
```

where the `fx` now returns the value of the i-th equality constraint evaluated at $x =$ `X` if `i` is specified. (This extension of the unconstrained case is best implemented by adding an optional argument `i` to the unconstrained version of `FUN`.) If derivatives are available, then the `GRADPROB` and (possibly) `HESSPROB` subroutines must be adapted as well

```
    GRADPROB( X, G, i )   HESSPROB( X, H, i )    ( i = 1,...,neq )
```

to compute the gradient and Hessian of the i-th constraint at $x =$ `X`. The constraints may be assigned names in the optional input argument

`CNAMES` (a size `neq` array of character fields of length 10)

The name of constraint i should appear in `CNAMES(i)` for each `i = 1, ..., neq`.

Note that, if the gradient of the objective function is provided, so must the gradients of the equality constraints. *The same level of derivative availability is assumed for all problem functions (objective and constraints).* The final values of the constraints and the values of their associated Lagrange multipliers are optionally returned to the user in the double precision output arguments `CX` and `Y`, respectively (both being of size `neq`).

2.4 Inequality constrained minimization problems

If inequality constraints are present in the problem, they are included just as for equality constraints. One needs to specify the optional input argument

`nin` (integer): the number q of inequality constraints.

The inequality constraints are then numbered from 1 to `nin` and their values or that of their derivatives is again computed by calling, for `i = 1,...,nin`,

```
FUNPROB( X, fx, i )  GRADPROB( X, G, i )  HESSPROB( X, H, i )
```

The inequality constraints are internally converted to equations by adding slack variables, whose names are set to `Slack_i`, where the character `i` in this string takes the integers values 1 to `nin`.) As in the equality-constrained case, the name of constraint `i` (`i = 1, ..., nin`) may be specified in `CNAMES(i)`. The values of the inequality constraints at the final `X` are returned (as for equalities) in the optional double precision output argument `CX` of size `nin`. The values of the corresponding Lagrange multipliers are returned in the optional double precision output argument `Y` of size `nin`.

2.5 Minimization problems with equality and inequality constraints

If there are both equalities and inequalities, `neq` and `nin` must be specified and the values and derivatives of the constraints are computed by

```
FUNPROB( X, fx, i )  GRADPROB( X, G, i )  HESSPROB( X, H, i ) (i=1,...,neq)
```

for the equality constraints, and

```
FUNPROB( X, fx, i )  GRADPROB( X, G, i )  HESSPROB( X, H, i ) (i=neq+1,neq+nin)
```

for the inequality constraints. The components `i = 1, ..., neq` of the vector `CNAMES` now contain the names of the `neq` equality constraints and components `i = neq + 1 , ..., neq + nin`, the names of the `nin` inequality constraints. Again, the same level of derivative availability is assumed for all problem functions (objective and constraints). Finally, the optional arguments `CX` and/or `Y`, if used, now have size `neq+nin`.

2.6 An example

Suppose we now consider the problem

$$\min_{x_1,x_2} f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \tag{2.1}$$

under the constraints

$$0 \leq x_1, \qquad x_1 + 3x_2 - 3 = 0, \qquad x_1^2 + x_2^2 - 4 \leq 0 \tag{2.2}$$

To solve this problem, we now use the call

```
    CALL LANCELOT_simple( 2, X, fx, exit_code, BL = (/0.0D0, -1.0D20/),  &
                          MY_FUN = FUN, MY_GRAD = GRAD, MY_HESS = HESS,  &
                          neq = 1, nin = 1, CX =  cx, Y = y )            &
```

(assuming we wish to examine `cx` and `y`), and modify the `FUN`, `GRAD` and `HESS` functions as follows

```
!.....................................................................
    SUBROUTINE FUN ( X, F, i )
!.....................................................................
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
    REAL( KIND = wp ), INTENT( IN )  :: X( : )
    REAL( KIND = wp ), INTENT( OUT ) :: F
    INTEGER, INTENT( IN ), OPTIONAL  :: i
    IF ( .NOT. PRESENT( i ) ) THEN
!       the objective function value (user defined)
!=====================================================================
        F = 100.0_wp*(X(2)-X(1)**2)**2 +(1.0_wp-X(1))**2                !
!=====================================================================
     ELSE
        SELECT CASE ( i )
            CASE ( 1 )
!               the equality constraint value (user defined)
!=====================================================================
                F = X(1)+3.0_wp*X(2)-3.0_wp                             !
!=====================================================================
            CASE ( 2 )
!               the inequality constraint value (user defined)
!=====================================================================
                F = X(1)**2+X(2)**2-4.0_wp                              !
!=====================================================================
        END SELECT
     END IF
     RETURN
    END SUBROUTINE FUN
!
!.....................................................................
    SUBROUTINE GRAD( X, G, i )
!.....................................................................
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
    REAL( KIND = wp ), INTENT( IN )  :: X( : )
    REAL( KIND = wp ), INTENT( OUT ) :: G( : )
    INTEGER, INTENT( IN ), OPTIONAL  :: i
    IF ( .NOT. PRESENT( i ) ) THEN
!       the objective functions's gradient components (user defined)
!=====================================================================
        G( 1 ) = -400.0_wp*(X(2)-X(1)**2)*X(1)-2.0_wp*(1.0_wp-X(1))     !
        G( 2 ) =  200.0_wp*(X(2)-X(1)**2)                               !
!=====================================================================
    ELSE
        SELECT CASE ( i )
            CASE ( 1 )
!               the equality constraint's gradient components (user defined)
!=====================================================================
                G( 1 ) =  1.0_wp                                        !
                G( 2 ) =  3.0_wp                                        !
!=====================================================================
            CASE ( 2 )
!               the inequality constraint's gradient components (user defined)
!=====================================================================
```

```
                G( 1 ) =  2.0_wp*X(1)                                          !
                G( 2 ) =  2.0_wp*X(2)                                          !
!================================================================================
        END SELECT
    END IF
    RETURN
    END SUBROUTINE GRAD
!
!.....................................................................
    SUBROUTINE HESS( X, H, i )
!.....................................................................
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
    REAL( KIND = wp ), INTENT( IN )  :: X( : )
    REAL( KIND = wp ), INTENT( OUT ) :: H( : )
    INTEGER, INTENT( IN ), OPTIONAL  :: i
    IF ( .NOT. PRESENT( i ) ) THEN
!       the entries of the upper triangle of the objective function's
!       Hessian  matrix,  stored by columns  (user defined)
!================================================================================
        H( 1 ) = -400.0_wp*(X(2)-3.0_wp*X(1)**2)+2.0_wp                        !
        H( 2 ) = -400.0_wp*X(1)                                                !
        H( 3 ) =  200.0_wp                                                     !
!================================================================================
    ELSE
        SELECT CASE ( i )
            CASE ( 1 )
!               the entries of the upper triangle of the equality
!               constraint's Hessian matrix, stored by columns (user defined)
!================================================================================
                H( 1 ) = 0.0_wp                                                !
                H( 2 ) = 0.0_wp                                                !
                H( 3 ) = 0.0_wp                                                !
!================================================================================
            CASE ( 2 )
!               the entries of the upper triangle of the inequality
!               constraint's Hessian matrix, stored by columns (user defined)
!================================================================================
                H( 1 ) = 2.0_wp                                                !
                H( 2 ) = 0.0_wp                                                !
                H( 3 ) = 2.0_wp                                                !
!================================================================================
        END SELECT
    END IF
    RETURN
    END SUBROUTINE HESS
```

Convergence is then obtained in 8 iterations. Note that, in our example, the objective function or its derivatives is/are computed if the index `i` is omitted.

3 Available algorithmic options

Beyond the choice of derivative level for the problem functions, the following optional arguments allow a (very limited) control of the algorithmic choices available in LANCELOT:

`maxit` **(integer):** maximum number of iterations (default if absent: 1000)

`gradtol` **(double precision):** the threshold on the maximum infinity norm of the gradient (or of the Lagrangian's gradient) required for declaring convergence (default if absent: 1.0d-5),

`feastol` **(double precision):** the threshold on the maximum infinity norm of the constraint violation required for declaring convergence (for constrained problems) (default if absent: 1.0d-5),

`print_level` **(integer):** a positive number proportional to the amount of output by the package: 0 corresponds to the silent mode, 1 to a single line of information per iteration (the default if absent), while higher values progressively produce more output.

Users should keep in mind that more elaborate algorithmic options are available when using more elaborate interfaces to the LANCELOT code within GALAHAD.

The full call corresponding to our constrained example could be

```
CALL LANCELOT_simple( 2, X, fx, exit_code,                          &
                      FUN = FUN, GRAD = GRAD, HESS = HESS,          &
                      BL  =  BL, BU = BU, VNAMES = VNAMES,          &
                      CNAMES = CNAMES,  neq = 1, nin = 1,           &
                      CX = cx, Y = y, ITERS = iters , maxit = 100,  &
                      gradtol = 1.0d-5, feastol = 1.0d-5,           &
                      print_level = 1 )
```

4 Compiling and linking

Of course, the interface described above is an interface within the GALAHAD library. Its use therefore requires the *a priori* installation of this library and the necessary mechanisms to access its content.

At compile time, the Fortran 90 compiler must be informed of the particular module where the interface can be found. Assuming GALAHAD is properly installed in double precision and that the environmental variable[3] `GALAHAD` has been set to point to the main GALAHAD directory (see Gould et al., 2003 for details), the relevant Fortran 90 module can be found in the directory

```
$GALAHAD/modules/pc.lnx.l95/double
```

where the string `pc.lnx.l95` indicates the particular architecture under which the GALAHAD library has been installed (in this case, a PC under Linux and the Lahay lf95 Fortran 90 compiler). This information needs to be provided to the compiler via an architecture-dependent flag (typically `-I`, `-M` or `-module`; see the makefile variable `MODULES` for your architecture in the directory `$GALAHAD/makefiles`).

Similarly, the linker must be able to include the relevant GALAHAD objects into the executable code. These objects are the GALAHAD library proper, as well as the BLAS, LAPACK, HSL and METIS libraries, which may all be found (in their public domain versions) in the directory

```
$GALAHAD/objects/pc.lnx.l95/double
```

The link command should therefore include a string of the form

```
-L$GALAHAD/objects/pc.lnx.l95/double -lgalahad -lgalahad_hsl \
 -lgalahad_metis -lgalahad_lapack -lgalahad_blas
```

for proper execution.

5 Single precision version

A single precision version is also available in the GALAHAD library. All real `LANCELOT_simple` should then be defined in single precision.

[3] in the Bourne or C-shell.

6 Other sources of information

The user is encouraged to consult the specsheet of the (non-naive) interface to LANCELOT within the GALAHAD software library for a better view of all possibilities offered by an intelligent use of the package. The library is described in Gould et al. (2003). The book by Conn, Gould and Toint (1992) is also a good source of additional information.

Acknowledgments

Many thanks to Anke Troeltzsch (CERFACS) for her most useful suggestions and to A. Neculai and L. Watson for motivating us to provide this interface.

References

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 *in* 'Springer Series in Computational Mathematics'. Springer Verlag, Heidelberg, Berlin, New York, 1992.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, **29**(4), 353–372, 2003.

Appendix: example calling code and makefile

In order to clarify the use of the interface as much as possible, we provide in this appendix the full Fortran 90 code of a calling program corresponding to our constrained example (2.1)-(2.2) of Section 2.6. The necessary subroutines `FUN`, `GRAD` and `HESS` are those of Section 2.6 and are assumed to be appended to calling program. We also give a makefile for compiling/linking this program into an executable file. These are intended as a base which can be modified to suit particular user needs.

The code of the calling program[4] is the following.

```
PROGRAM RUN_LANCELOT_simple

!-------------------------------------------------------------------------------
!                                                                              !
!    This programs provides a simple (naive) way to run LANCELOT B on an       !
!    optimization problem without interaction with CUTEr and using a simple    !
!    representation of the problem where the objective function and             !
!    constraints comprise each a single group with a single nonlinear          !
!    element.                                                                  !
!                                                                              !
!    The use of this program and the accompanying script lanb_simple           !
!    requires that the GALAHAD package has prealably be installed.            !
!                                                                              !
!    Ph. L. Toint, November 2007.                                              !
!    Copyright reserved, Gould/Orban/Toint, for GALAHAD productions            !
!                                                                              !
!-------------------------------------------------------------------------------
!
    USE LANCELOT_simple_double
!
    IMPLICIT NONE
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
    REAL ( KIND = wp ), PARAMETER :: infinity = 10.0_wp ** 20
```

[4] This code is available in the GALAHAD installation tree in the file ./src/lanb_simple/run_lancelot_simple.f90.

```
    INTEGER :: n, neq, nin, iters, maxit, print_level, exit_code
    REAL ( KIND = wp ) :: gradtol, feastol, fx
    CHARACTER ( LEN = 10 ), ALLOCATABLE, DIMENSION(:) :: VNAMES, CNAMES
    REAL ( KIND = wp ),     ALLOCATABLE, DIMENSION(:) :: BL, BU, X, CX, Y
    EXTERNAL :: FUN, GRAD, HESS
!
! THE TEST PROBLEM DIMENSIONS (user defined)
!
    n   = 2  ! number of variables
    neq = 1  ! number of equality constraints, excluding fixed variables
    nin = 1  ! number of inequality (<= 0) constraints, excluding bounds
!
! allocate space for problem defining vectors
!
    ALLOCATE( X( n  ), BL( n ), BU( n ), CX( neq+nin ), Y( neq+nin ) )
    ALLOCATE( VNAMES( n  ),CNAMES( neq+nin ) )
!
! starting point
!
    X(1) = -1.2_wp                ! starting point (componentwise)
    X(2) =  1.0_wp
!
! bounds on the variables
!
    BL(1) =  0.0_wp               ! lower bounds (componentwise)
    BL(2) = -infinity
    BU(1) =  infinity             ! upper bounds (componentwise)
    BU(2) =  3.0_wp
!
! names
!
    VNAMES(1) = 'x1'              ! variables
    VNAMES(2) = 'x2'
    CNAMES(1) = 'Equality'        ! equality constraints
    CNAMES(2) = 'Inequality'      ! inequality constraints
!
! algorithmic parameters
!
    maxit       = 100
    gradtol     = 0.00001_wp      ! identical to default
    feastol     = 0.00001_wp      ! identical to default
    print_level = 1               ! identical to default
!
! solve by calling LANCELOT B
!
    CALL LANCELOT_simple( n,  X, fx, exit_code,                          &
                          MY_FUN = FUN, MY_GRAD = GRAD , MY_HESS = HESS, &
                          BL = BL, BU = BU, VNAMES   =  VNAMES,          &
                          CNAMES =  CNAMES, neq = neq, nin = nin,        &
                          CX = CX, Y = Y, ITERS  = iters, MAXIT = maxit, &
                          GRADTOL = gradtol, FEASTOL = feastol,          &
                          PRINT_LEVEL = print_level )
!
! clean up
!
    DEALLOCATE( X, BL, BU, CX, Y )
    DEALLOCATE( VNAMES, CNAMES )
    STOP
```

```
!
END PROGRAM RUN_LANCELOT_simple
```

The makefile[5] is then given for the specific default `DOUBLE PRECSION` case where

- the Lahay lf95 Fortran compiler is used,
- the architecture specified at the installation of the GALAHAD library is pc.linx.l95 (running on a PC under linux with lf95),
- the environment variable `$GALAHAD` has been set properly (as recommended in the installation of the GALAHAD library).

```
#######################################################################
#                                                                     #
# Makefile                                                            #
#                                                                     #
#    compiles and links the simple 1-element naive interface          #
#    to LANCELOT B, with the program RUN_LANCELOT_simple.f90.         #
#    This makefile requires the prior installation of the             #
#    GALAHAD package, including the definition of the GALAHAD         #
#    environment variable (This variable needs to point to the main   #
#    directory in which the GALAHAD package has been installed.)      #
#                                                                     #
#    D. Orban, December 2007.                                         #
#    Copyright reserved, Gould/Orban/Toint, for GALAHAD productions   #
#                                                                     #
#######################################################################

FORTRAN = lf95
ARCH    = pc.lnx.195
PREC    = double

MAIN    = run_lancelot_simple
USRLIBS =

GALMODULES = ${GALAHAD}/modules/$(ARCH)/$(PREC)
GALLIBDIR  = ${GALAHAD}/objects/$(ARCH)/$(PREC)
GALLIBS1   = -lgalahad -lgalahad_blas -lgalahad_hsl
GALLIBS2   = -lgalahad_metis -lgalahad_lapack
GALLIBS    = -L$(GALLIBDIR) $(GALLIBS1) $(GALLIBS2)

%.o: %.f90
        $(FORTRAN) -c -I$(GALMODULES) $+

$(MAIN): $(MAIN).o
        $(FORTRAN) -o $@ $+ $(GALLIBS) $(USRLIBS)
```

Modifying this makefile for another example, another compiler or another precision requires (possibly) adapting the definitions of the name of the calling program (here `run_lancelot_naive.f90`), the variables `MAIN`, `USRLIBS`, `FORTRAN`, `ARCH` and `PREC`. The options in the compiling and linking commands may also vary for other Fortran 90 compilers. This can be achieved by issuing commands of the type

```
make FORTRAN=g95 ARCH=mac.osx.g95 PREC=single MAIN=spamalot USRLIBS=-lg2c
```

In our example (which corresponds to the default settings in the makefile), the simple commands

```
make
run_lancelot_simple
```

produce the following output:

[5] This makefile is available in the GALAHAD installation tree in the file ./src/lanb_simple/bin/Makefile.

```
            ********************************************
            *                                          *
            *              LANCELOT_simple             *
            *                                          *
            *  a simple interface to GALAHAD/LANCELOT B  *
            *                                          *
            ********************************************

Penalty parameter   1.0000E-01 Required projected gradient norm =   1.0000E-01
                               Required constraint         norm =   1.0000E-01

*********  Starting optimization  **************

There are        3 variables
There are        3 groups
There are        3 nonlinear elements

Objective function value    1.01000000000000E+02
Constraint norm             2.00000000000000E+00

 Iter #g.ev c.g.it      f    proj.g    rho   radius   step   cgend #free   time
    0     1     0  1.21E+02 1.6E+02     -       -       -      -       3    0.0
    1     2     1  3.83E+01 5.8E+01  8.9E-01 1.0E+00 1.0E+00 -CURV     3    0.0
    2     3     2  1.01E+02 3.0E+00  9.8E-02 1.0E+00 1.0E+00 BOUND     3    0.0
    3     4     3  7.71E+00 2.1E+00  1.1E+00 1.0E+00 8.7E-01 CONVR     3    0.0
    4     5     4  4.85E-01 2.2E+00  1.1E+00 1.7E+00 5.2E-01 CONVR     3    0.0
    5     6     5  2.73E-02 8.5E-01  1.1E+00 1.7E+00 9.9E-02 CONVR     3    0.0
    6     7     6  2.32E-02 2.1E-02  1.1E+00 1.7E+00 1.1E-02 CONVR     3    0.0

Iteration number              6  Merit function value    =   2.31890590071E-02
No. derivative evals          7  Projected gradient norm =   2.08773257903E-02
C.G. iterations               6  Trust-region radius     =   1.73866572139E+00
Number of updates skipped     0

There are      3 variables and      0 active bounds

Times for Cauchy, systems, products and updates    0.00    0.00    0.00    0.00

Exact Cauchy step computed
Bandsolver preconditioned C.G. ( semi-bandwidth =      5 ) used
Infinity-norm trust region used
Non-monotone descent strategy ( history =  1 ) used
Exact second derivatives used

Objective function value    2.30556773462821E-02

Penalty parameter       =   1.0000E-01
Projected gradient norm =   2.0877E-02 Required gradient   norm =   1.0000E-01
Constraint         norm =   5.1647E-03 Required constraint norm =   1.0000E-01

******** Updating multiplier estimates **********

Penalty parameter   1.0000E-01 Required projected gradient norm =   1.0000E-02
                               Required constraint         norm =   1.2589E-02

 Iter #g.ev c.g.it      f    proj.g    rho   radius   step   cgend #free   time
    6     7     6  2.36E-02 1.5E-01     -       -       -      -       3    0.0
    7     8     7  2.34E-02 3.4E-04  1.0E+00 1.0E+00 3.6E-03 CONVR     3    0.0

Iteration number              7  Merit function value    =   2.34466960383E-02
No. derivative evals          8  Projected gradient norm =   3.42123908435E-04
C.G. iterations               7  Trust-region radius     =   1.00000000000E+00
Number of updates skipped     0

There are      3 variables and      0 active bounds

Times for Cauchy, systems, products and updates    0.00    0.00    0.00    0.00

Exact Cauchy step computed
Bandsolver preconditioned C.G. ( semi-bandwidth =      5 ) used
```

```
Infinity-norm trust region used
Non-monotone descent strategy ( history =  1 ) used
Exact second derivatives used

Objective function value    2.33210093681760E-02

Penalty parameter       =   1.0000E-01
Projected gradient norm =   3.4212E-04 Required gradient   norm =   1.0000E-02
Constraint         norm =   1.5101E-04 Required constraint norm =   1.2589E-02

******** Updating multiplier estimates **********

Penalty parameter   1.0000E-01 Required projected gradient norm =   1.0000E-03
                               Required constraint         norm =   1.5849E-03

 Iter #g.ev c.g.it      f    proj.g    rho    radius   step   cgend #free   time
    7     8     7  2.34E-02 5.2E-03     -       -       -       -       3    0.0
    8     9     8  2.34E-02 2.5E-07  1.0E+00 1.0E+00 6.2E-05 CONVR     3    0.0

Iteration number              8  Merit function value    =   2.34391260921E-02
No. derivative evals          9  Projected gradient norm =   2.45864887574E-07
C.G. iterations               8  Trust-region radius     =   1.00000000000E+00
Number of updates skipped     0

There are      3 variables and      0 active bounds

Times for Cauchy, systems, products and updates    0.00    0.00    0.00    0.00

Exact Cauchy step computed
Bandsolver preconditioned C.G. ( semi-bandwidth =      5 ) used
Infinity-norm trust region used
Non-monotone descent strategy ( history =  1 ) used
Exact second derivatives used

Objective function value    2.33135045070784E-02

Penalty parameter       =   1.0000E-01
Projected gradient norm =   2.4586E-07 Required gradient   norm =   1.0000E-03
Constraint         norm =   2.9716E-06 Required constraint norm =   1.5849E-03

Variable name Number Status     Value   Lower bound Upper bound  |  Dual value
------------- ------ ------     -----   ----------- -----------  |  ----------
 x1              1     FREE  8.4750E-01  0.0000E+00  1.0000E+20  |  2.4586E-07
 x2              2     FREE  7.1750E-01 -1.0000E+20  3.0000E+00  | -5.0502E-08
 Slack_1         3     FREE  2.7669E+00  0.0000E+00  1.0000E+20  |  2.3343E-08

Constraint name Number    Value    Scale factor | Lagrange multiplier
--------------- ------    -----    ----- ------ | -------------------
   Equality        1   -1.2974E-06  1.0000E+00  |     5.0124E-02
   Inequality      2   -2.9716E-06  1.0000E+00  |     2.3343E-08

Objective function value    2.33135045070784E-02

There are       3 variables in total.
There are       2 equality constraints.
Of these        1 are primal degenerate.
There are       0 variables on their bounds.
Of these        0 are dual degenerate.
```