


GALAHAD LANCELOT B

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

1 SUMMARY

LANCELOT B is a suite of Fortran procedures for minimizing an objective function, where the minimization variables are required to satisfy a set of auxiliary, possibly nonlinear, constraints. The objective function is represented as a sum of "group" functions. Each group function may be a linear or nonlinear functional whose argument is the sum of "finite element" functions; each finite element function is assumed to involve only a few variables or have a second derivative matrix with low rank for other reasons. The constraints are also represented as group functions. Bounds on the variables and known values may be specified. The routine is especially effective on problems involving a large number of variables.

1.1 A brief introduction to terminology and scope

The objective function has the (so-called *group-partial separable*) form

$$f(\mathbf{x}) = \sum_{i \in \mathcal{G}_O} w_i^g g_i \left(\sum_{j \in \mathcal{E}_i} w_{ij}^e e_j(\mathbf{x}_j^e, \mathbf{p}_j^e) + \mathbf{a}_i^T \mathbf{x} - b_i, \mathbf{p}_i^g \right), \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T, \tag{1.1}$$

where $\mathcal{G}_O$ is a subset of $\mathcal{G} = \{1, \ldots, n_g\}$, the list of indices of group functions g_i, each $\mathcal{E}_i$ is a subset of $\mathcal{E} = \{1, \ldots, n_e\}$, the list of indices of nonlinear element functions e_j, and the w_i^g, $\mathbf{p}_i^g$, w_{ij}^e and $\mathbf{p}_i^e$ are, respectively, group and element weights and vectors of parameters. Furthermore, the vectors $\mathbf{x}_j^e$ are either small subsets of the minimization variables $\mathbf{x}$ or are such that the rank of the second derivative matrix of the nonlinear element e_j is small for some other reason, and the vectors $\mathbf{a}_i$ of the linear elements are sparse. The least value of the objective function within the intersection of the regions defined by the general constraints

$$c_i(\mathbf{x}) = w_i^g g_i \left(\sum_{j \in \mathcal{E}_i} w_{ij}^e e_j(\mathbf{x}_j^e, \mathbf{p}_j^e) + \mathbf{a}_i^T \mathbf{x} - b_i, \mathbf{p}_i^g \right) = 0, \quad i \in \mathcal{G}_C = \mathcal{G} \setminus \mathcal{G}_O \setminus \mathcal{G}_I, \tag{1.2}$$

and the "box"

$$l_j \leq x_j \leq u_j, \quad 1 \leq j \leq n,$$

is sough; here $\mathcal{G}_I$ is another subset of $\mathcal{G}$ that may be used to specify a list of groups to be (temporarily) ignored. Either bound on each variable may be infinite, and special considerations are taken when there are no general constraints.

The method used is iterative. There are two levels of iteration. In the outer, a composite function, the (augmented Lagrangian) merit function,

$$\phi(\mathbf{x}, \mathbf{y}, \mu) = f(\mathbf{x}) + \sum_{i \in \mathcal{G}_C} y_i c_i(\mathbf{x}) + \frac{1}{2\mu} \sum_{i \in \mathcal{G}_C} (c_i(\mathbf{x}))^2, \tag{1.3}$$

where μ is known as the penalty parameter and $\mathbf{y}$ is a vector of Lagrange multiplier estimates, is formulated. Each outer iteration requires the approximate minimization of this merit function within the feasible box, for given values of μ and $\mathbf{y}$.

The required approximate minimization for fixed μ and $\mathbf{y}$ is carried out using a series of inner iterations. At each inner iteration, a quadratic model of the merit function is constructed. An approximation to the minimizer of this model within a trust-region is calculated. The trust region can be either a "box" or a hypersphere of specified radius, centered at the current best estimate of the minimizer. If there is an accurate agreement between the model and the true objective function at the new approximation to the minimizer, this approximation becomes the new best estimate.

Otherwise, the radius of the trust region is reduced and a new approximate minimizer sought. The algorithm also allows the trust-region radius to increase when necessary. The minimization of the model function can be carried out by using either a direct-matrix or an iterative approach.

The approximate minimization of the model is performed in two stages. In the first, a so-called generalized Cauchy point is determined by approximately minimizing the model within the intersection of the feasible box and the trust-region along a scaled steepest descent direction. Having taken this step, the model is further reduced by solving one or more quadratic minimization problems in which any constraints activated at the Cauchy point remain so. The latter computation is essentially equivalent to the solution of a sequence of linear systems, and may be performed using either direct (factorization) or iterative (preconditioned conjugate gradient) method.

After an appropriate approximation to the minimizer of the merit function is obtained, and if there are general constraints, μ and $\mathbf{y}$ are adjusted to ensure convergence of the outer iteration to the required solution of the constrained minimization problem.

If there is a linear transformation of variables such that one or more of the element functions depend on fewer transformed "internal" variables than the original number of variables $\mathbf{x}_i^e$, this may be specified. This can lead to a substantial reduction in computing time.

There are facilities for the user to provide a preconditioner for the conjugate gradient solution of the internal linear systems (a default is provided) and to provide (optional) first and second derivatives for (subsets) of the element functions. The subroutine optionally returns control to the user to calculate function and derivative values of the group functions g_i and nonlinear element functions e_j. A large number of other options are available, such as the ability to choose between different generalized Cauchy points, to select different trust-region shapes, including a structured trust region especially designed for functions with structure like those in (1.1)–(1.2), and to permit non-monotonic descent of the merit function. Some of these options require external packages, and thus are not available by default (see Section 1.2).

1.2 External packages

Some of the linear system options make use of the HSL package `MA27`. This package is available without charge for academic purposes from the HSL archive

```
http://hsl.rl.ac.uk/archive/hslarchive.html .
```

Another option, the Lin-Moré preconditioner `ICFS`, is available as part of the MINPACK 2 library, via

```
http://www-unix.mcs.anl.gov/~more/icfs/ .
```

Munksgaard's preconditioner `MA61` is only available commercially from HSL. See

```
http://www.cse.clrc.ac.uk/Activity/HSL+152
```

for details. In addition, the GALAHAD package `SILS` may be replaced by the functionally equivalent but state-of-the-art HSL package `HSL_MA57` from the same source.

ATTRIBUTES — Versions: `LANCELOT_single, LANCELOT_double`, **Uses:** `GALAHAD_CPU_time, GALAHAD_SPECFILE, LANCELOT_INITW, LANCELOT_OTHERS, LANCELOT_HSPRD, LANCELOT_CAUCHY, LANCELOT_CG, LANCELOT_PRECN, LANCELOT_FRNTL, LANCELOT_STRUTR, GALAHAD_SMT, GALAHAD_SILS, GALAHAD_SCU, LANCELOT_ASMBL, LANCELOT_EXTEND`. **Date:** February 2002. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, England, and Ph. L. Toint, University of Namur, Belgium. Previous Fortran 77 version also by A. R. Conn, IBM T. J. Watson Research Center, USA. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Remark:** This is an enhanced version of the Fortran 77 subroutines `SBMIN` and `AUGLG`, available as part of LANCELOT A, and supersedes them.

2 HOW TO USE THE PACKAGE

The package is available using both single and double precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE LANCELOT_single
```

with the obvious substitution `LANCELOT_double`, `LANCELOT_single_64` and `LANCELOT_double_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `LANCELOT_problem_type`, `LANCELOT_time_type`, `LANCELOT_control_type`, `LANCELOT_inform_type` (Section 2.2) and the subroutines `LANCELOT_initialize`, `LANCELOT_solve`, `LANCELOT_terminate`, (Section 2.3) and `LANCELOT_read_specfile` (Section 2.6) must be renamed on one of the `USE` statements.

2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions and `DOUBLE PRECISION` for the double precision cases, and correspond to `rp_ = real32` and `rp_ = real64`, respectively, as supplied by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.2 The derived data types

Five derived data types are accessible from the package.

2.2.1 The derived data type for holding the problem

The derived data type `LANCELOT_problem_type` is used to hold the problem data. The components of `LANCELOT_problem_type` are:

`n` is a scalar variable of type `INTEGER(ip_)`, that holds the number of optimization variables, n.

`ng` is a scalar variable of type `INTEGER(ip_)`, that holds the number of group functions, n_g.

`nel` is a scalar variable of type `INTEGER(ip_)`, that holds the number of nonlinear element functions, n_e.

`IELING` is a rank-one allocatable array of dimension `ISTADG(ng+1)-1` and type `INTEGER(ip_)`, that holds the indices of the nonlinear elements $\mathcal{E}_i$ used by each group. The indices for group i must immediately precede those for group $i+1$ and each group's indices must appear in a contiguous list. See Section 5 for an example.

`ISTADG` is a rank-one allocatable array of dimension `ng+1` and type `INTEGER(ip_)`, whose i-th value gives the position in `IELING` of the first nonlinear element in group function i. In addition, `ISTADG(ng+1)` should be equal to the position in `IELING` of the last nonlinear element in group `ng` plus one. See Table 2.1 and Section 5 for an example.

`IELVAR` is a rank-one allocatable array of dimension `ISTAEV(nel+1)-1` and type `INTEGER(ip_)`, that holds the indices of the variables in the first nonlinear element e_1, followed by those in the second nonlinear element e_2, ... See Section 5 for an example.

`ISTAEV` is a rank-one allocatable array of dimension `nel+1` and type `INTEGER(ip_)`, whose k-th value is the position of the first variable of the k-th nonlinear element function, in the list `IELVAR`. In addition, `ISTAEV(`n_e`+1)` must be equal to the position of the last variable of element n_e in `IELVAR` plus one. See Table 2.2 and Section 5 for an example.

Table 2.1: Contents of the arrays IELING, ESCALE and ISTADG

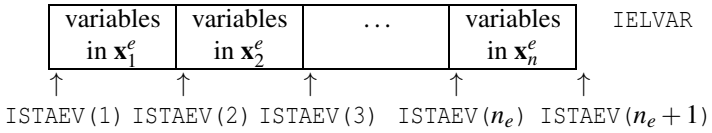

Table 2.2: Contents of the arrays IELVAR and ISTAEV

INTVAR is a rank-one allocatable array of dimension nel+1 and type INTEGER(ip_), whose i-th value must be set to the number of internal variables required for the i-th nonlinear element function e_i on initial entry. See Section 2.4.3 for the precise definition of internal variables and Section 5 for an example.

ISTADH is a rank-one allocatable array of dimension nel+1 and type INTEGER(ip_), that is set in LANCELOT_solve so that its i-th value $(1 \leq i \leq n_e)$ gives the position in the array FUVALS (see Sections 2.3.2 and 2.4.4, and (particularly) Tables 2.6 and 2.7) of the first component of the Hessian matrix of the i-th nonlinear element function e_i, *with respect to its internal variables*. Only the upper triangular part of each Hessian matrix is stored and the storage is by columns (see Section 2.4.7 for details). The element ISTADH(n_e+1) gives the position in FUVALS of the first component of the gradient of the objective function.

ICNA is a rank-one allocatable array of dimension ISTADA(ng+1)-1 and type INTEGER(ip_), that holds the indices of the nonzero components of a_1, the gradient of the first linear element, in any order, followed by those in a_2, etc. See Table 2.3 and Section 5 for an example.

ISTADA is a rank-one allocatable array of dimension ng+1 and type INTEGER(ip_), whose i-th value is the position of the first nonzero component of the i-th linear element gradient, a_i, in the list ICNA. In addition, ISTADA(ng+1) must be equal to the position of the last nonzero component of a_{n_g} in ICNA plus one. See Table 2.3 and Section 5 for an example.

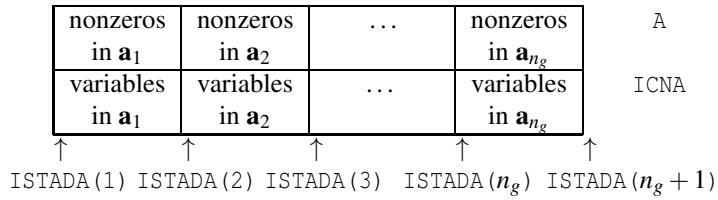

Table 2.3: Contents of the arrays A, ICNA and ISTADA

KNDOFG is a rank-one allocatable array of dimension ng and type INTEGER(ip_), that is used to indicate which of the groups are to be included in the objective function, which define equality constraints, and which are to be ignored. If KNDOFG is not ASSOCIATED, all groups will be included in the objective function, and it will be assumed that there are no general constraints. If KNDOFG is ASSOCIATED, each of the first n_g entries of KNDOFG

must be set to 0, 1 or 2 on initial entry. If KNDOFG(*i*) has the value 1, *i* lies in the set $\mathcal{G}_O$ and the group will be included in the objective function (1.1). If, KNDOFG(*i*) has the value 2, *i* lies in the complement set $\mathcal{G}_C$, and the *i*-th group defines an equality constraint of the form (1.2). Finally, if KNDOFG(*i*) has the value 0, group *i* will be ignored; this is useful when, for example, many optimizations are required with different subsets of constraints, or when a feasible point is sought without reference to the objective.

ITYPEE is a rank-one allocatable array of dimension nel and type INTEGER(ip_), that is used to indicate the types (See Section 2.4.2) of the nonlinear element functions. The *i*-th component of ITYPEE specifies the type of element *i*.

ISTEPA is a rank-one allocatable array of dimension nel + 1 and type INTEGER(ip_), whose *i*-th component gives the position in EPVALU (see below) of the first parameter for element function *i*. In addition, ISTEPA(nel+1) is the position in EPVALU of the last parameter for element function n_e plus one. See Table 2.4 for an illustration.

Table 2.4: Contents of the arrays EPVALU and ISTEPA

ITYPEG is a rank-one allocatable array of dimension ng and type INTEGER(ip_), that is used to indicate the types (See Section 2.4.2) of the group functions. The *i*-th component of ITYPEG specifies the type of group *i*.

ISTGPA is a rank-one allocatable array of dimension ng +1 and type INTEGER(ip_), whose *i*-th component gives the position in GPVALU (see below) of the first parameter for group function *i*. In addition, ISTGPA(ng+1) is the position in GPVALU of the last parameter for element function n_g plus one. See Table 2.5 for an illustration.

Table 2.5: Contents of the arrays GPVALU and ISTGPA

A is a rank-one array of dimension ISTADA(ng+1)-1 and type REAL(rp_), that holds the values of the nonzero components of the gradients of the linear element functions, $\mathbf{a}_i$, $i = 1,\ldots,n_g$. The values must appear in the same order as their indices appear in ICNA, i.e., the nonzero from element *i*, whose index is, say, ICNA(*k*) will have value A(*k*). See Table 2.3 and Section 5 for an example.

B is a rank-one array of dimension ng and type REAL(rp_), whose *i*-th entry holds the value of the constant b_i for each group.

BL is a rank-one array of dimension n and type REAL(rp_), whose *i*-th entry must be set to the value of the lower bound l_i on the *i*-th variable. If the *i*-th variable has no lower bound, BL(*i*) should be set to a large negative number.

BU is a rank-one array of dimension n and type REAL(rp_), whose *i*-th entry must be set to the value of the upper bound u_i on the *i*-th variable. If the *i*-th variable has no upper bound, BU(*i*) should be set to a large positive number.

X is a rank-one array of dimension n and type REAL(rp_), that holds the current values of the minimization variables, $\mathbf{x}$.

C is a rank-one array of dimension ng and type REAL(rp_), that holds the current estimates of the values of the equality constraints for the problem. If KNDOFG(*i*) < 2, C(*i*) will not be set, but if KNDOFG(*i*) $= 2$, C(*i*) contains the constraint value $c_i(\mathbf{x})$ of the *i*-th constraint (1.2). C need not be ASSOCIATED if KNDOFG is not.

Y is a rank-one array of dimension ng and type REAL(rp_), that holds the current estimates of the Lagrange multipliers, $\mathbf{y}$, for the problem. If KNDOFG(*i*) < 2, Y(*i*) will not be set, but if KNDOFG(*i*) $= 2$, Y(*i*) contains the multiplier estimate y_i for the *i*-th constraint (1.2). Y need not be ASSOCIATED if KNDOFG is not.

GSCALE is a rank-one array of dimension ng and type REAL(rp_), whose *i*-th entry holds the value of *i*-th group weight w_i^g.

ESCALE is a rank-one array of dimension ISTADG(ng+1)-1 and type REAL(rp_), whose entries hold the values of element weights w_{ij}^e. The weights must occur in the same order as the indices of the elements assigned to each group in IELING, with the weights for the elements in group *i* preceding those in group $i+1$, $i = 1,\ldots,$ng-1. See Table 2.1 and Section 5 for an example.

VSCALE is a rank-one array of dimension n and type REAL(rp_), that holds suitable positive scale factors for the problem variables $\mathbf{x}$. The i-th variable x_i will implicitly be divided by VSCALE(*i*) within LANCELOT_solve. The scale factors should ideally be chosen so that the rescaled variables are of order one at the solution to the minimization problem. If the user does not know suitable scalings, each component of VSCALE should be set to 1.0. Good variable scalings can result in considerable savings in computing times.

EPVALU is a rank-one array of dimension ISTEPA(nel+1)-1 and type REAL(rp_), that holds the values of the element parameters $\mathbf{p}_i^e$, $i = 1,\ldots,n_e$. The indices for the parameters for element *i* immediately precede those for element $i+1$, and each element's parameters appear in a contiguous list. See Table 2.4 for an illustration.

GPVALU is a rank-one array of dimension ISTGPA(ng+1)-1 and type REAL(rp_), that holds the values of the group parameters $\mathbf{p}_i^g$, $i = 1,\ldots,n_g$. The indices for the parameters for group *i* immediately precede those for group $i+1$, and each group's parameters appear in a contiguous list. See Table 2.5 for an illustration.

GXEQX is a rank-one array of dimension ng and type default LOGICAL, whose *i*-th entry must be set .TRUE. if the *i*-th group function is the trivial function $g(x) = x$ (see Section 2.4.2) and .FALSE. otherwise.

INTREP is a rank-one array of dimension nel and type default LOGICAL, whose *i*-th entry must be set .TRUE. if the *i*-th nonlinear element function has a useful transformation between elemental and internal variables and .FALSE. otherwise (see Section 2.4.3).

VNAMES is a rank-one array of dimension n and type default CHARACTER and length 10, whose *j*-th entry contains the "name" of the *j*-th variable.

GNAMES is a rank-one array of dimension ng and type default CHARACTER and length 10, whose *i*-th entry contains the "name" of the *i*-th group.

2.2.2 The derived data type for holding control parameters

The derived data type LANCELOT_control_type is used to hold controlling data. Default values may be obtained by calling LANCELOT_initialize (see Section 2.3.1), while components may also be changed by calling LANCELOT_read_spec (see Section 2.6.1). The components of LANCELOT_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in LANCELOT_solve and LANCELOT_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in LANCELOT_solve is suppressed if out ≤ 0. The default is out $= 6$.

alive_unit is a scalar variable of type INTEGER(ip_). If alive_unit $>$ 0, a temporary file named alive_file (see below) will be created on stream number alive_unit on initial entry to LANCELOT_solve, and execution of LANCELOT_solve will continue so long as this file continues to exist. Thus, a user may terminate execution simply by removing the temporary file from this unit. If alive_unit $\leq$ 0, no temporary file will be created, and execution cannot be terminated in this way. The default is alive_unit $=$ 60.

alive_file is a scalar variable of type default CHARACTER and length 30, that gives the name of the temporary file whose removal from stream number alive_unit terminates execution of LANCELOT_solve. The default is alive_unit $=$ ALIVE.d.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if tt print_level $\leq$ 0. If print_level $=$ 1, a single line of output will be produced for each iteration of the process, while additionally if print_level $=$ 2 a summary of the inner iteration will be given. If print_level $\geq$ 3, this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

maxit is a scalar variable of type INTEGER(ip_), that holds the maximum number of iterations which will be allowed in LANCELOT_solve. The default is maxit = 1000.

start_print is a scalar variable of type INTEGER(ip_), that specifies the first iteration for which printing will be permitted in LANCELOT_solve. If start_print is negative, printing will be permitted from the outset. The default is start_print = -1.

stop_print is a scalar variable of type INTEGER(ip_), that specifies the last iteration for which printing will be permitted in LANCELOT_solve. If stop_print is negative, printing will be permitted once it has been started by start_print. The default is stop_print = -1.

print_gap is a scalar variable of type INTEGER(ip_). Once printing has been started, output will occur once every print_gap iterations. If print_gap is no larger than 1, printing will be permitted on every iteration. The default is print_gap = 1.

linear_solver is a scalar variable of type INTEGER(ip_), that is used to specify the method used to solve the linear systems of equations which arise at each iteration of the minimization algorithm. Possible values are:

1. The conjugate gradient method will be used without preconditioning.
2. A preconditioned conjugate gradient method will be used with a diagonal preconditioner.
3. A preconditioned conjugate gradient method will be used with a user supplied preconditioner. Control will be passed back to the user to construct the product of the preconditioner with a given vector prior to a reentry to LANCELOT_solve (see Section 2.4.7).
4. A preconditioned conjugate gradient method will be used with an expanding band incomplete Cholesky preconditioner. This option is only available if the user has provided one of the external packages MA27 or HSL_MA57, (see Section 1.2).
5. A preconditioned conjugate gradient method will be used with Munksgaard's preconditioner. This option is only available if the user has provided the external package MA61 (see Section 1.2).
6. A preconditioned conjugate gradient method will be used with a modified Cholesky preconditioner due to Gill, Murray, Ponceléon and Saunders, in which an indefinite factorization is altered to give a positive definite one. This option is only available if the user has provided one of the external packages MA27 or HSL_MA57, (see Section 1.2).
7. A preconditioned conjugate gradient method will be used with a modified Cholesky preconditioner due to Schnabel and Eskow, in which small or negative diagonals are made sensibly positive during the factorization. This option is only available if the user has provided one of the external packages MA27 or HSL_MA57, (see Section 1.2).

8. A preconditioned conjugate gradient method will be used with a band preconditioner. The semi-bandwidth is given by the variable semibandwidth (see below).

9. A preconditioned conjugate gradient method will be used with an incomplete Cholesky factorization preconditioner due to Lin and Moré. This option is only available if the user has provided the external package ICFS, (see Section 1.2).

11. A multifrontal factorization method will be used. This option is only available if the user has provided one of the external packages MA27 or HSL_MA57, (see Section 1.2).

12. A modified Cholesky factorization method will be used. This option is only available if the user has provided one of the external packages MA27 or HSL_MA57, (see Section 1.2).

Any other value of linear_solver will be regarded as 1. The default is linear_solver = 8.

icfact is a scalar variable of type INTEGER(ip_), that specifies the number of extra vectors of length prob%n used by the Lin and Moré preconditioner (linear_solver = 4) if requested. Usually, the larger the number, the better the preconditioner, but the more space and effort required to use it. Any negative value will be regarded as 0. The default is icfact = 5.

semibandwidth is a scalar variable of type INTEGER(ip_), that holds the semi-bandwidth used if the band preconditioner (linear_solver = 8) is selected. Any negative value will be regarded as 0. The default is semibandwidth = 5.

max_sc is a scalar variable of type INTEGER(ip_), that specifies the maximum number of variables that are allowed to hit their bounds during the calculation of the step before a refactorization of the preconditioner is triggered. A dense Schur-complement matrix of order at most max_sc may be generated in lieu of the refactorization, so some compromise between many refactorizations and the storage of a potentially large, dense Schur complement must be made. Any non-positive value will be regarded as 1. The default is max_sc = 75.

io_buffer is a scalar variable of type INTEGER(ip_), that holds the unit number of an input/output buffer for writing temporary files during array-size re-allocations, if needed. The default is io_buffer = 75.

more_toraldo is a scalar variable of type INTEGER(ip_), that specifies the number of Moré-Toraldo projected searches that are to be used to improve upon the Cauchy point when finding the step. Any non-positive value results in a standard add-one-at-a-time conjugate-gradient search. The default is more_toraldo = 0.

non_monotone is a scalar variable of type INTEGER(ip_), that specifies the history-length for non-monotone descent strategy. Any non-positive value results in standard monotone descent, for which merit function improvement occurs at each iteration. There are often definite advantages in using a non-monotone strategy with a modest history, since the occasional local increase in the merit function may enable the algorithm to move across (gentle) "ripples" in the merit function surface. However, we do not usually recommend large values of non_monotone. The default is non_monotone = 1.

first_derivatives is a scalar variable of type INTEGER(ip_), that specifies what sort of first derivative approximation to use. If the user is able to provide analytical first derivatives for each nonlinear element function, first_derivatives must be set to to be non-positive. If analytical first derivatives are unavailable, they may be estimated by forward differences by setting first_derivatives = 1, or, more accurately but at additional expense, by central differences by setting first_derivatives $\geq$ 2. The default is first_derivatives = 0. **N.B.** This value will be ignored whenever the user provides the optional argument ELDERS to LANCELOT_solve.

second_derivatives is a scalar variable of type INTEGER(ip_), that specifies what sort of second derivative approximation to use. If the user is able to provide analytical second derivatives for each nonlinear element function, second_derivatives must be set to be non-positive. If the user is unable to provide second derivatives, these derivatives will be approximated using one of four secant approximation formulae. If second_derivatives is set to 1, the BFGS formula is used; if it is set to 2, the DFP formula is used; if it is set to 3, the PSB formula is

used; and if it is set to 4 or larger, the symmetric rank-one formula is used. The user is strongly advised to use exact second derivatives if at all possible as this often significantly improves the convergence of the method. The default is `second_derivatives = 0`. **N.B.** This value will be ignored whenever the user provides the optional argument `ELDERS` to `LANCELOT_solve`.

`stopg` is a scalar variable of type `REAL(rp_)`, that is used to specify the maximum permitted (infinity) norm of the projected gradient of the Lagrangian function (see Section 4) at the estimate of the solution sought. The default is `stopg` $= 10^{-5}$.

`stopc` is a scalar variable of type `REAL(rp_)`, that is used to specify the maximum permitted violation (measured in infinity norm) of the constraints at the estimate of the solution sought. The default is `stopc` $= 10^{-5}$.

`min_aug` is a scalar variable of type `REAL(rp_)`, that is used to specify the smallest permitted value of the merit function (1.3). Any smaller value will result in the minimization being terminated. The default is `min_aug = -HUGE(1.0)/8.0`.

`acccg` is a scalar variable of type `REAL(rp_)`, that is used to specify the least relative reduction in (two-) norm of the residual of the reduced gradient of the model function that is required from the conjugate gradient iteration. The default is `acccg = 0.01`.

`initial_radius` is a scalar variable of type `REAL(rp_)`, that holds the required initial value of the trust-region radius. If `initial_radius` ≤ 0, the radius will be chosen automatically by `LANCELOT_solve` The default is `initial_radius = - 1.0`.

`maximum_radius` is a scalar variable of type `REAL(rp_)`, that holds the maximum permitted value of the trust-region radius. Any radius that exceeds the larger of `maximum_radius` and one during the calculation will be reset to `MAX(maximum_radius,1.0)`. The default is `maximum_radius` $= 10^{20}$.

`eta_successful`, `eta_very_successful` and `eta_extremely_successful` are scalar variables of type default `REAL(rp_)`, that control the acceptance and rejection of the trial step and the updates to the trust-region radius. At every iteration, the ratio of the actual reduction in the merit function following the trial step to that predicted by the model is computed. The step is accepted whenever this ratio exceeds `eta_successful`; otherwise the trust-region radius will be reduced. If, in addition, the ratio exceeds `eta_very_successful`, the trust-region radius may be increased. If a structured trust-region is being used (see `structured_tr` below), the radius for an individual element will only be increased if additionally the ratio of the actual to predicted decrease for this element exceeds `eta_extremely_successful`. The defaults are `eta_successful = 0.01`, `eta_very_successful = 0.9` and `eta_extremely_successful = 0.95`.

`gamma_smallest`, `gamma_decrease` and `gamma_increase` are scalar variables of type `REAL(rp_)`, that control the maximum amounts by which the trust-region radius can contract or expand during an iteration. The radius will be decreased by powers of `gamma_decrease` until it is smaller than the larger of an internally calculated prediction of what should be a good value and `gamma_smallest`. It can be increased by at most a factor `gamma_increase`. The defaults are `gamma_smallest = 0.0625`, `gamma_decrease = 0.25` and `gamma_increase = 2.0`.

`mu_meaningful_model` and `mu_meaningful_group` are scalar variables of type `REAL(rp_)`, that hold tolerances that determine whether a group (and its model) is "meaningful" when a structured trust-region is being used (see `structured_tr` below). A model of an individual group is meaningful if the change in its value that results from the trial step is more than a factor `mu_meaningful_model` of the overall model decrease. The same is true of the group itself if if the change in its value that results from the trial step is more than a factor `mu_meaningful_group` of the overall model decrease. The radius update strategies are more liberal for non-meaningful groups (see Conn, Gould, Toint, 2000, Section 10.2.2). These values are for experts only. The defaults are `mu_meaningful_model = 0.01` and `mu_meaningful_group = 0.1`.

initial_mu is a scalar variable of type REAL(rp_), that specifies the initial value of the penalty parameter, μ. Any value smaller than u will be replaced by u, where u is EPSILON(1.0) (EPSILON(1.0D0) in LANCELOT_double). The default is initial_mu = 0.1.

mu_tol is a scalar variable of type REAL(rp_), that holds a threshold value of the penalty parameter, above which no attempt will be made to update Lagrange multiplier estimates. The default is mu_tol = 0.1.

firstg is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted (infinity) norm of the projected gradient of the Lagrangian function (see Section 4) at the end of the first major iteration. The default is firstg = 0.1.

firstc is a scalar variable of type REAL(rp_), that is used to specify the maximum hoped-for violation (measured in infinity norm) of the constraints at the end of the first major iteration. No attempt will be made to update Lagrange multiplier estimates until the violation falls below this threshold. The default is firstc = 0.1.

cpu_time_limit is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted CPU time. Any negative value indicates no limit will be imposed. The default is cpu_time_limit = - 1.0.

quadratic_problem is a scalar variable of type default LOGICAL, that should be set .TRUE. if the objective function is quadratic (or linear) and any constraints linear, and .FALSE. otherwise. LANCELOT B can take some advantage of quadratic problems, although the GALAHAD packages GALAHAD_QPB or GALAHAD_QPA are to be preferred in this case. The default is quadratic_problem = .FALSE..

two_norm_tr is a scalar variable of type default LOGICAL, that should be set .TRUE. if a two-norm (hyperspherical) trust region is required, and .FALSE. if an infinity-norm (box) trust region is to be used. The default is two_norm_tr = .FALSE..

exact_gcp is a scalar variable of type default LOGICAL, that should be set .TRUE. if the exact generalized Cauchy point, the first estimate of the minimizer of the quadratic model within the feasible box, is required, and .FALSE. if an approximation suffices. The default is exact_gcp = .TRUE..

accurate_bqp is a scalar variable of type default LOGICAL, that should be set .TRUE. if an accurate minimizer of the quadratic model within the feasible box is required, and .FALSE. if an approximation suffices. The accurate minimizer often requires considerably more work, but occasionally this reduces the overall number of iterations. The default is accurate_bqp = .FALSE..

structured_tr is a scalar variable of type default LOGICAL, that should be set .TRUE. if the shape of the trust region should be adjusted to account for the partial separability of the problem, and .FALSE. otherwise. The default is structured_tr = .FALSE..

print_max is a scalar variable of type default LOGICAL, that should be set .TRUE. if the *printed* values of the objective function and its derivatives should be multiplied by minus one (as might be the case if we aim to maximize $f(\mathbf{x})$ by minimizing $-f(\mathbf{x})$), and .FALSE. otherwise. The default is print_max = .FALSE..

full_solution is a scalar variable of type default LOGICAL, that should be set .TRUE. if all components of the solution $\mathbf{x}$ and constraints $\mathbf{c}(\mathbf{x})$ should be printed on termination when print_level > 0, and .FALSE. if only the first and last (representative) few are required. The default is full_solution = .TRUE..

space_critical is a scalar variable of type default LOGICAL, that must be set .TRUE. if space is critical when allocating arrays and .FALSE. otherwise. The package may run faster if space_critical is .FALSE. but at the possible expense of a larger storage requirement. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

SILS_cntl is a scalar variable of type SILS_control, that holds control parameters for the GALAHAD sparse matrix factorization package SILS. See the documentation for the package SILS for details of the derived type SILS_control, and of the default values given to its components.

2.2.3 The derived data type for holding problem data

The derived data type LANCELOT_data_type is used to hold all the data for a particular problem between calls of LANCELOT procedures. This data should be preserved, untouched, from the initial call to LANCELOT_initialize to the final call to LANCELOT_terminate.

2.2.4 The derived data type for holding informational parameters

The derived data type LANCELOT_info_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of LANCELOT_info_type are:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Sections 2.4.7 and 2.5 for details. A value status = 0 indicates successful termination.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or de-allocation.

iter is a scalar variable of type INTEGER(ip_), that gives the number of iterations that have been performed since the start of the minimization. This is one fewer than the number of problem function evaluations that have been made.

itercg is a scalar variable of type INTEGER(ip_), that gives the number of conjugate-gradient iterations that have been performed since the start of the minimization.

itcgmx is a scalar variable of type INTEGER(ip_), that gives the maximum number of conjugate-gradient iterations that will be allowed at each iteration of the inner minimization.

ncalcf is a scalar variable of type INTEGER(ip_), that gives the number of element functions that must be re-evaluated when the reverse-communication mode is used to evaluate element values.

ncalcg is a scalar variable of type INTEGER(ip_), that gives the number of group functions that must be re-evaluated when the reverse-communication mode is used to evaluate group values.

nvar is a scalar variable of type INTEGER(ip_), that gives the current number of free variables, that is the number of variables which do not lie on either simple bounds.

ngeval is a scalar variable of type INTEGER(ip_), that gives the number of problem function gradient evaluations that have been made since the start of the minimization.

iskip is a scalar variable of type INTEGER(ip_), that gives the number of times that an approximation to the second derivatives of an element function has been rejected.

ifixed is a scalar variable of type INTEGER(ip_), that gives the index of the variable that most recently encountered one of its bounds in the minimization process.

nsemib is a scalar variable of type INTEGER(ip_), that gives the bandwidth used if the band or expanding band preconditioner is selected. (control%linear_solver = 4 or 8).

aug is a scalar variable of type REAL(rp_), that gives the current value of the augmented Lagrangian merit function.

obj is a scalar variable of type REAL(rp_), that gives the current value of the objective function.

pjgnrm is a scalar variable of type REAL(rp_), that gives the (infinity) norm of the current projected gradient of the Lagrangian function (see Section 4).

cnorm is a scalar variable of type REAL(rp_), that gives the (infinity) norm of the current violation of the constraints.

ratio is a scalar variable of type REAL(rp_), that gives the ratio of the actual reduction that has been made in the merit function value during the current iteration to that predicted by the model function. A value close to one is to be expected as the algorithm converges.

mu is a scalar variable of type REAL(rp_), that gives the current value of the penalty parameter.

radius is a scalar variable of type REAL(rp_), that gives the current value of the radius of the trust-region.

ciccg is a scalar variable of type REAL(rp_), that gives the current value of the pivot tolerance that is used when Munksgaard's incomplete Cholesky factorization (control%linear_solver = 5) is chosen for preconditioning.

newsol is a scalar variable of type default LOGICAL, that will be .TRUE. if a major iteration has just been completed, and .FALSE. otherwise.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last array for which an array allocation/de-allocation was unsuccessful.

SCU_info is a scalar variable of type SCU_info_type, that holds informational parameters concerning the the GALAHAD bordered matrix factorization package SCU. See the documentation for the package SCU for details of the derived type SCU_info_type.

SILS_infoa is a scalar variable of type SILS_ainfo, that holds informational parameters concerning the analysis subroutine contained in the GALAHAD sparse matrix factorization package SILS. See the documentation for the package SILS for details of the derived type SILS_ainfo.

SILS_infof is a scalar variable of type SILS_finfo, that holds informational parameters concerning the factorization subroutine contained in the GALAHAD sparse matrix factorization package SILS. See the documentation for the package SILS for details of the derived type SILS_finfo.

2.3 Argument lists and calling sequences

There are three principal procedures for user calls (see Section 2.6 for further features):

1. The subroutine LANCELOT_initialize is used to set default values, and initialize private data, before solving the specified optimization problem.

2. The subroutine LANCELOT_solve is called to solve the optimization problem.

3. The subroutine LANCELOT_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by LANCELOT_solve, at the end of the solution process.

We use square brackets [ ] to indicate OPTIONAL arguments.

2.3.1 The initialization subroutine

Default values are provided as follows:

```
CALL LANCELOT_initialize( data, control )
```

data is a scalar INTENT(INOUT) argument of type LANCELOT_data_type (see Section 2.2.3). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type LANCELOT_control_type (see Section 2.2.2). On exit, control contains default values for the components as described in Section 2.2.2. These values should only be changed after calling LANCELOT_initialize.

2.3.2 The minimization subroutine

The minimization algorithm is called as follows:

```
CALL LANCELOT_solve( prob, RANGE, GVALS, FT, XT, FUVALS, lfuval, ICALCF, ICALCG, &
                     IVAR, Q, DGRAD, control, inform, data[, ELDERS] [, ELFUN]   &
                     [, ELFUN_flexible] [, GROUP] )
```

prob is a scalar INTENT(INOUT) argument of type LANCELOT_problem_type (see Section 2.2.1). It is used to hold data about the problem being solved. Components n, ng, nel, IELING, ISTADG, IELVAR, ISTAEV, ICNA, ISTADA, A, B, BL, BU, GSCALE, ESCALE, VSCALE, GXEQX, INTREP, VNAMES, and GNAMES must all be set on entry as described in Section 2.2.1, and will thereafter be unaltered.

The component INTVAR must also be set as described in Section 2.2.1, but will subsequently be reset by LANCELOT_solve so that its *i*-th value gives the position in the array FUVALS (see below) of the first component of the gradient of the *i*-th nonlinear element function, *with respect to its internal variables* (see FUVALS). The component X must also be set, but will be altered by LANCELOT_solve. The component ISTADH need not be set on initial entry, but will be set by LANCELOT_solve as described in Section 2.2.1.

If the dummy argument ELFUN (see below) is present, the components ISTEPA and EPVALU must also be set on entry as described in Section 2.2.1—they will not subsequently be altered—but otherwise they need not be associated or assigned. Likewise if the dummy argument GROUP (see below) is present, the components ISTGPA, GPVALU and ITYPEG must also be set on entry as described in Section 2.2.1—they will not subsequently be altered—but otherwise they need not be associated or assigned.

If the problem involves general constraints (1.2), the components KNDOFG, Y and C must be associated, and the first two assigned values—KNDOFG will not subsequently be altered, while C will be set by LANCELOT_solve. If the problem does not involve general constraints, KNDOFG, Y and C need not be associated.

RANGE is a user-supplied subroutine whose purpose is to define the linear transformation of variables for those nonlinear elements which have different elemental and internal variables. See Section 2.4.3 for details. RANGE must be declared EXTERNAL in the calling program.

GVALS is a rank-two INTENT(OUT) array argument of shape (prob%ng, 3) and type REAL(rp_) that is used to store function and derivative information for the group functions. The user may be asked to provide values for these functions and/or derivatives, evaluated at the argument FT (see FT) when control is returned to the calling program with a negative value of the variable inform%status. This information needs to be stored by the user in specified locations within GVALS. Details of the required information are given in Section 2.4.7.

FT is a rank-one INTENT(INOUT) array argument of dimension prob%ng and type REAL(rp_) that is set within LANCELOT_solve to a trial value of the argument of the *i*-th group function at which the user may be required to evaluate the values and/or derivatives of that function. Precisely what group function information is required at FT is under the control of the variable inform%status and details are given in Section 2.4.7.

XT is a rank-one INTENT(INOUT) array argument of dimension prob%n and type REAL(rp_) that is set within LANCELOT_solve to a trial value of the variables $\mathbf{x}$ at which the user may be required to evaluate the values and/or derivatives of the nonlinear elements functions. Precisely what element function information is required at XT is under the control of the variable inform%status and details are given in Section 2.4.7.

FUVALS is a rank-one INTENT(INOUT) array argument of dimension lfuval and type REAL(rp_) that is used to store function and derivative information for the nonlinear element functions. The user is asked to provide values for these functions and/or derivatives, evaluated at the argument XT (see XT), at specified locations within FUVALS, when control is returned to the calling program with a negative value of the variable inform%status. Alternatively, the user may have provided a suitable subroutine ELFUN to compute the required function or derivative values (see below). Details of the required information are given in Sections 2.4.4 and 2.4.7. The layout of FUVALS is indicated in Table 2.6.

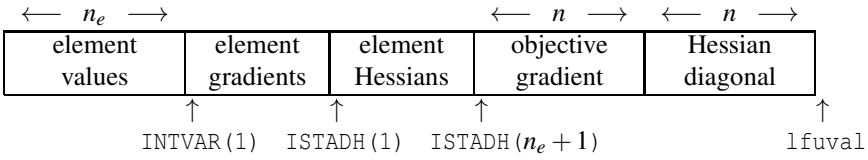

Table 2.6: Partitioning of the workspace array FUVALS

The first segment of FUVALS contains the values of the nonlinear element functions; the next two segments contain their gradients and Hessian matrices, taken with respect to their internal variables, as indicated in Section 2.4.7. The remaining two used segments contain the gradient of the objective function and the diagonal elements of the second derivative approximation, respectively. At the solution, the components of the gradient of the augmented Lagrangian function corresponding to variables which lie on one of their bounds are of particular interest in many applications areas. In particular they are often called shadow prices, and are used to assess the sensitivity of the solution to variations in the bounds on the variables.

lfuval is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the actual length of FUVALS in the calling program. The required length may be calculated using the following code segment:

```
lfuval = prob%nel + 2 * prob%n
DO i = 1, prob%nel
  lfuval = lfuval + ( prob%INTVAR( i ) * ( prob%INTVAR( i ) + 3 ) ) / 2
END DO
```

ICALCF is a rank-one INTENT(INOUT) array argument of dimension prob%nel and type INTEGER(ip_). If the user has chosen not to perform internal element evaluations, and if the value of inform%status on return from LANCELOT_solve indicates that further element functions values or their derivatives are required prior to a re-entry, the first inform%ncalcf components of ICALCF give the indices of the group functions which need to be recalculated at XT. Precisely what group function information is required is under the control of the variable inform%status and details are given in Section 2.4.7.

ICALCG is a rank-one INTENT(INOUT) array argument of dimension prob%ng and type INTEGER(ip_). If the user has chosen not to perform internal group evaluations, and if the value of inform%status on return from LANCELOT_solve indicates that further group functions values or their derivatives are required prior to a re-entry, the first inform%ncalcg components of ICALCG give the indices of the group functions which need to be recalculated at FT. Precisely what group function information is required is under the control of the variable inform%status and details are given in Section 2.4.7.

IVAR is a rank-one INTENT(INOUT) array argument of dimension prob%n and type INTEGER(ip_) that is required when the user is providing a special preconditioner for the conjugate gradient inner iteration (see Section 2.4.7).

Q and DGRAD are rank-one INTENT(INOUT) array arguments of dimension prob%n and type REAL(rp_) that are required when the user is providing a special preconditioner for the conjugate gradient inner iteration (see Section 2.4.7).

control is a scalar INTENT(INOUT) argument of type LANCELOT_control_type (see Section 2.2.2). On exit, control contains default values for the components as described in Section 2.2.2. These values should only be changed after calling LANCELOT_initialize.

inform is a scalar INTENT(INOUT) argument of type LANCELOT_type (see Section 2.2.4). The component status must be set to 0 on initial entry, and a successful call to LANCELOT_solve is indicated when the component status has the value 0. For other return values of status, see Sections 2.4.7 and 2.5.

data is a scalar INTENT(INOUT) argument of type LANCELOT_data_type (see Section 2.2.3). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to LANCELOT_initialize.

ELDERS is an OPTIONAL rank-two INTENT(INOUT) array argument of shape (2, prob%nel) and type INTEGER(ip_) that may be used to specify what kind of first- and second-derivative approximations will be required for each nonlinear element function. If ELDERS is not present, the first- and second-derivative requirements for every element will be as specified by the variables control%first_derivatives and control%second_derivatives (see Section 2.2.2) respectively. For finer control, if ELDERS is present, and the user is able to provide analytical first derivatives for the *i*-th nonlinear element function, ELDERS(1,i) must be set to to be non-positive. If analytical first derivatives are unavailable, they may be estimated by forward differences by setting ELDERS(1,i) = 1 or, more accurately but at additional expense, by central differences by setting ELDERS(1,i) $\geq$ 2. Similarly, if ELDERS is present, and the user is able to provide analytical second derivatives for the *i*-th nonlinear element function, ELDERS(2,i) must be set to to be non-positive. If the user is unable to provide second derivatives, these derivatives will be approximated using one of four secant approximation formulae. If ELDERS(2,i) is set to 1, the BFGS formula is used; if it is set to 2, the DFP formula is used; if it is set to 3, the PSB formula is used; and if it is set to 4 or larger, the symmetric rank-one formula is used. The user is strongly advised to use analytic first and second derivatives if at all possible as this often significantly improves the convergence of the method.

ELFUN and ELFUN_flexible are OPTIONAL user-supplied subroutines whose purpose is to evaluate the values and derivatives of the nonlinear element functions. See Section 2.4.1 for background information and Sections 2.4.4 and 2.4.5 for details. Only one of ELFUN and ELFUN_flexible may be present at once, and, if present, must be declared EXTERNAL in the calling program. Which of the two arguments is permitted depends on whether ELDERS (see above) is present. ELFUN_flexible is only permitted when ELDERS is present, since ELFUN_flexible allows the user to provide element-specific levels of derivative information. In the absence of ELDERS, ELFUN should be used instead of ELFUN_flexible. If both ELFUN and ELFUN_flexible are absent, LANCELOT_solve will use reverse communication to obtain element function values and derivatives.

GROUP is an OPTIONAL user-supplied subroutine whose purpose is to evaluate the values and derivatives of the group functions. See Section 2.4.1 for background information and Section 2.4.6 for details. If GROUP is present, it must be declared EXTERNAL in the calling program. If GROUP is absent, LANCELOT_solve will use reverse communication to obtain group function values and derivatives. GROUP need not be present if all components of prob%GXEQX are .TRUE..

2.3.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL LANCELOT_terminate( data, control, inform )
```

`data` is a scalar `INTENT(INOUT)` argument of type `LANCELOT_data_type` exactly as for `LANCELOT_solve` that must not have been altered **by the user** since the last call to `LANCELOT_initialize`.

`control` is a scalar `INTENT(IN)` argument of type `LANCELOT_control_type` exactly as for `LANCELOT_solve`.

`inform` is a scalar `INTENT(INOUT)` argument of type `LANCELOT_type` exactly as for `LANCELOT_solve`. Only the components `status`, `alloc_status` and `bad_alloc` might have been altered on exit, and a successful call to `LANCELOT_terminate` is indicated when this component `status` has the value 0. For other return values of `status`, see Section 2.5.

2.4 Function and derivative values

2.4.1 Evaluating problem functions

As we saw in Section 1.1, both the objective function (1.1) and constraints (1.2) are composed from element functions $e_j(\mathbf{x}_j^e, \mathbf{p}_j^e)$ and group functions $g_i(\alpha_i, \mathbf{p}_i^g)$, all other defining data being linear (and thus described accurately on initial input to `LANCELOT_solve` by components of `prob`)—here we refer to the *scalar*

$$\alpha_i \stackrel{\text{def}}{=} \sum_{j \in \mathcal{E}_i} w_{ij}^e e_j(\mathbf{x}_j^e, \mathbf{p}_j^e) + \mathbf{a}_i^T \mathbf{x} - b_i.$$

as the i-th *group variable*. If we wish to evaluate (1.1)–(1.2) and their derivatives, it thus remains for the user to supply function and derivatives of the element and group functions to `LANCELOT_solve` as required. `LANCELOT_solve` allows the user to do this in one of two ways.

The first method simply requires that the user provides subroutines, with prescribed argument lists, that accept input values $(\mathbf{x}_j^e, \mathbf{p}_j^e)$ or $(\alpha_i, \mathbf{p}_i^g)$, and provide as output $e_j(\mathbf{x}_j^e, \mathbf{p}_j^e)$ or $g_i(\alpha_i, \mathbf{p}_i^g)$ or their derivatives. If this method is used, control will only return to the user once the required minimizer has been obtained (or an error condition flagged). The disadvantage of this approach is the inflexibility of the argument lists, while the advantage is often its speed of execution.

The second method is to exit from `LANCELOT_solve` whenever function or derivative values are required, and simply ask the user to provide values/derivatives of $e_j(\mathbf{x}_j^e, \mathbf{p}_j^e)$ or $g_i(\alpha_i, \mathbf{p}_i^g)$ from given arguments $(\mathbf{x}_j^e, \mathbf{p}_j^e)$ or $(\alpha_i, \mathbf{p}_i^g)$ before re-entering. Here, the advantage is its flexibility—function/derivative values may be computed by any means, possibly even by a completely separate calculation—while the disadvantage is there may be severe performance penalties as argument values are saved and retrieved on entry/exit from `LANCELOT_solve`.

The first method is known as *internal evaluation*, while the second is an example of *reverse communication*. Note that we allow the possibility that the user may wish to provide element values/derivatives using one of the above options, and group values/derivatives using the other.

2.4.2 Element and group function types

The value of the i-th element function e_j depends on its *elemental* variables $\mathbf{x}_j$ and its parameters $\mathbf{p}_j^e$. It is extremely common in practice that although each element has its own list of elemental variables and parameters, the functional form of many (if not all) is identical. For example, suppose that $\mathbf{x}_j = (x_j)$ and $\mathbf{p}_j^e = (p_j)$, and that

$$e_j(\mathbf{x}_j, \mathbf{p}_j^e) = p_j x_j^2.$$

Then each element $e_j(\mathbf{x}_j, \mathbf{p}_j^e$ can be expressed as

$$e_j(\mathbf{x}_j, \mathbf{p}_j^e) \equiv e(x_j, p_j)$$

of the single element *type*

$$e(x, p) = p x^2.$$

Thus, rather than writing a subroutine to evaluate the values and derivatives of *each* element separately, we simply need to write a subroutine to evaluate the value/derivative of the *single* element type, and then to evaluate these values/derivatives at each pair (x_i, p_i) in turn.

This very simple idea can significantly simplify the coding of function/derivative evaluation subroutines. In practice, it is unlikely that each element is of the same type, but usually a few types suffice to describe all the elements. Thus we provide the flexibility of allowing the user of specifying how many element types are involved, and of saying to which type each element belongs. Traditionally, if there are n_e^t types, they will be numbered from 1 to n_e^t, and there will be a mapping $e^t : \mathcal{E} \to \{1,\ldots,n_e^t\}$ that describes the type for each element $i \in \mathcal{E}$. In practice, the types may be numbered in whatever way best suits the user, so long as distinct types have unique identifiers—in this case, the set of type identifiers is some $\mathcal{E}^t$, and $e^t : \mathcal{E} \to \mathcal{E}^t$. Moreover, there is nothing to prevent the user from assigning each element its own type (in which case $n_e^t = n_e$), and setting $e_t(i) = i$.

Precisely the same idea applies to group functions. We will allow the user to specify how many group types, n_g^t there are, and to provide a mapping $g^t : \mathcal{G} \to \mathcal{G}^t$, the set of group type identifiers. It is also convenient to have a special group type, the *trivial* group, for which $g(\alpha) = \alpha$, since this special type is extremely common—the value/derivative of any group identified as trivial is indeed so trivial to compute that the user is not required to do so!

2.4.3 Internal variables and Subroutine `RANGE`

A nonlinear element function e_j is assumed to be a function of the variables $\mathbf{x}_j^e$, a subset of the problem variables $\mathbf{x}$. Suppose that $\mathbf{x}_j^e$ has n_j elements. Then another way of saying this is that (ignoring any element parameters) we have an element function $e_j(v_1,\ldots,v_{n_j})$, where in our special case, we choose $v_1 = (x_j^e)_1, \ldots, v_{n_j} = (x_j^e)_{n_j}$. The *elemental* variables for the element function e_j are the variables $\bar{v}$ and, while we need to associate the particular values x_j^e with $\bar{v}$ (using the array `IELVAR`), it is the elemental variables which are important in defining the nonlinear element functions.

As an example, the seventh nonlinear element function for a particular problem might be

$$e_7(v_1,v_2,v_3) = (v_1+v_2)e^{v_1-v_3}, \tag{2.1}$$

where for our example $v_1 = x_{29}$, $v_2 = x_3$ and $v_3 = x_{17}$. For this example, there are three elemental variables. However, the example illustrates an additional point. Although e_7 is a function of three variables, the function itself is really only composed of *two* independent parts; the product of (v_1+v_2) with $e^{v_1-v_3}$, or, if we write $u_1 = v_1+v_2$ and $u_2 = v_1-v_3$, the product of u_1 with e^{u_2}. The variables u_1 and u_2 are known as *internal* variables for the element function. They are obtained as *linear combinations* of the elemental variables. The important feature as far as `LANCELOT_solve` is concerned is that each nonlinear function involves as few variables as possible, as this allows for compact storage and more efficient derivative approximation. By representing the function in terms of its internal variables, this goal is achieved. `LANCELOT_solve` only stores derivatives of the element functions with respect to internal variables, so it pays to use an internal representation in this case. It frequently happens, however, that a function does not have useful internal variables. For instance, another element function might be

$$e_9(v_1,v_2) = v_1 \sin v_2, \tag{2.2}$$

where for example $v_1 = x_6$ and $v_2 = x_{12}$. Here, we have broken e_9 down into as few pieces as possible. Although there are internal variables, $u_1 = v_1$ and $u_2 = v_2$, they are the same in this case as the elemental variables and there is no virtue in exploiting them. Moreover it can happen that although there are special internal variables, there are just as many internal as elemental variables and it therefore doesn't particularly help to exploit them. For instance, if

$$e_{14}(v_1,v_2) = (v_1+v_2)\log(v_1-v_2),$$

where for example $v_1 = x_{12}$ and $v_2 = x_2$, the function can be formed as $u_1\log(u_2)$ where $u_1 = v_1+v_2$ and $u_2 = v_1-v_2$. But as there are just as many internal variables as elementals, it will not normally be advantageous to use this internal representation.

Finally, although an element function may have useful internal variables, the user need not bother with them. `LANCELOT_solve` will still work, but at the expense of extra storage and computational effort. The user decides on

input to LANCELOT_solve which elements have useful transformations by setting the appropriate elements of the array INTREP to .TRUE. .

In general, there will be a linear transformation from the elemental variables to the internal ones. For example (2.1), we have

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix},$$

while in (2.2), we have

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

Most generally, the transformation will be of the form

$$\mathbf{u} = \mathbf{W}\mathbf{v},$$

and this transformation is *useful* if the matrix $\mathbf{W}$ has fewer rows than columns.

The purpose of the RANGE routine is to define the transformation between internal and elemental variables for nonlinear elements with useful internal representations. The routine has the following argument list:

```
SUBROUTINE RANGE( ielemn, transp, W1, W2, nelvar, ninvar, ieltyp, lw1, lw2 )
```

ielemn is a scalar INTENT(IN) argument of type INTEGER(ip_), that gives the index of the nonlinear element whose transformation is required by LANCELOT_solve.

transp is a scalar INTENT(IN) argument of type default LOGICAL. If transp is .FALSE., the subroutine must put the result of the transformation $\mathbf{W}\mathbf{v}$ in the array W2, where $\mathbf{v}$ is input in the array W1. Otherwise, the subroutine must supply the result of the transposed transformation $\mathbf{W}^T\mathbf{u}$ in the array W2, where $\mathbf{u}$ is input in the array W1.

W1 is a rank-one INTENT(IN) array argument of type REAL(rp_) whose dimension is the number of elemental variables if transp is .FALSE. and the number of internal variables otherwise.

W2 is a rank-one INTENT(OUT) array argument of type REAL(rp_) whose dimension is the number of internal variables if transp is .FALSE. and the number of elemental variables otherwise. The result of the transformation of W1 or its transpose, as defined by transp, must be set in W2.

nelvar is a scalar INTENT(IN) argument of type INTEGER(ip_), that gives the number of elemental variables for the element specified by ielemn.

ninvar is a scalar INTENT(IN) argument of type INTEGER(ip_), which gives the number of internal variables for the element specified by ielemn.

ieltyp is a scalar INTENT(IN) argument of type INTEGER(ip_), that defines the type for the element specified by ielemn.

lw1 is a scalar INTENT(IN) argument of type INTEGER(ip_), that will have been set to the actual length of W1.

Restriction: lw1 $\geq$ ninvar if transp is .TRUE. and lw1 $\geq$ nelvar if transp is .FALSE. .

lw2 is a scalar INTENT(IN) argument of type INTEGER(ip_), that will have been set to the actual length of W2.

Restriction: lw2 $\geq$ nelvar if transp is .TRUE. and lw2 $\geq$ ninvar if transp is .FALSE. .

The user will already have specified which elements have useful transformations in the array INTREP. RANGE will only be called for elements for which the corresponding component of INTREP is .TRUE. .

2.4.4 Element function values and derivatives via internal evaluation

If the argument ELFUN is present when calling LANCELOT_solve, the user is expected to provide a subroutine of that name to evaluate function or derivative values (with respect to internal variables) of (a given subset of) the element functions. **N.B.** This routine is only permitted if the argument ELDERS is absent. The routine has the following argument list:

```
SUBROUTINE ELFUN ( FUVALS, XVALUE, EPVALU, ncalcf, ITYPEE, ISTAEV, IELVAR, INTVAR, &
                   ISTADH, ISTEPA, ICALCF, ltypee, lstaev, lelvar, lntvar, lstadh, &
                   lstepa, lcalcf, lfuval, lxvalu, lepvlu, ifflag, ifstat )
```

FUVALS is a rank-one INTENT(INOUT) array argument of dimension lfuval and type REAL(rp_) which is used to store function and derivative information for the nonlinear element functions. The subroutine is asked to provide values for these functions and/or derivatives, evaluated at the argument XVALUE (see XVALUE), at specified locations within FUVALS. The layout of FUVALS is indicated in Table 2.7.

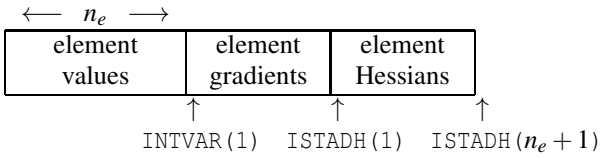

Table 2.7: Partitioning of the workspace array FUVALS

The first segment of FUVALS is used to hold the values of the nonlinear element functions, component i holding the value of the i-th element function. The second segment holds the components of the gradients of the element functions taken with respect to their internal variables, as described by INTVAR below. The final segment contains the elements' Hessian matrices, again taken with respect to their internal variables, as described by ISTADH below.

XVALUE is a rank-one INTENT(IN) array argument of dimension lxvalu and type REAL(rp_) that contains the values of $\mathbf{x}$ at which the subroutine is required to evaluate the values or derivatives of the nonlinear elements functions.

EPVALU is a rank-one INTENT(IN) array argument of dimension lepvlu and type REAL(rp_) that contains the values of the element parameters $\mathbf{p}_i^e$, $i = 1,\ldots,n_e$. The indices for the parameters for element i immediately precede those for element $i+1$, and each element's parameters appear in a contiguous list. See Table 2.4 for an illustration.

ncalcf is a scalar INTENT(IN) argument of type INTEGER(ip_), that specifies how many of the nonlinear element functions or their derivatives are to be evaluated.

ITYPEE is a rank-one INTENT(IN) array argument of dimension ltypee and type INTEGER(ip_) whose i-th component specifies the type of element i.

ISTAEV is a rank-one INTENT(IN) array argument of dimension lstaev and type INTEGER(ip_) exactly as described in Section 2.2.1.

IELVAR is a rank-one INTENT(IN) array argument of dimension lelvar and type INTEGER(ip_) exactly as described in Section 2.2.1.

INTVAR is a rank-one INTENT(IN) array argument of dimension lntvar and type INTEGER(ip_) whose i-th component ($1 \leq i \leq n_e$) gives the position in the array FUVALS of the first component of the gradient of the i-th nonlinear element function, *with respect to its internal variables*. See Table 2.8 for an illustration.

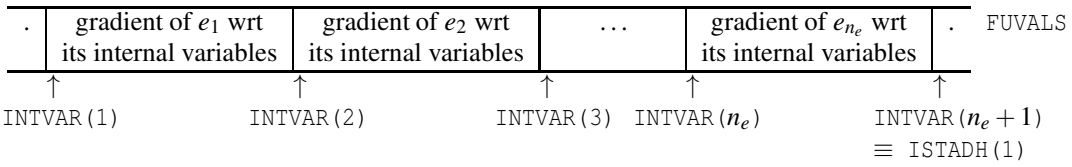

Table 2.8: Contents of the arrays INTVAR and (part of) FUVALS

ISTADH is a rank-one INTENT(IN) array argument of dimension lstadh and type INTEGER(ip_) whose i-th component $(1 \leq i \leq n_e)$ gives the position in the array FUVALS of the first component of the Hessian matrix of the i-th nonlinear element function e_i, *with respect to its internal variables*. Only the upper triangular part of each Hessian matrix is stored and the storage is by columns. That is to say that the component of the Hessian of the k-th nonlinear element with respect to internal variables i and j, $i \leq j$,

$$\frac{\partial^2 e_k}{\partial u_i \partial u_j},$$

must be placed in FUVALS(ISTADH(k)+$(j(j-1)/2)+i-1$). The element ISTADH(n_e+1) is space required to finish storing the Hessian of the last nonlinear element in FUVALS plus one. See Table 2.9 for an illustration.

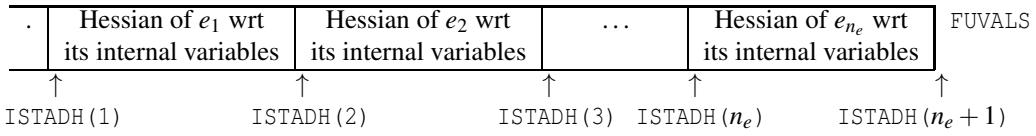

Table 2.9: Contents of the arrays ISTADH and (part of) FUVALS

ISTEPA is a rank-one INTENT(IN) array argument of dimension lstepa and type INTEGER(ip_) whose i-th component gives the position in EPVALU of the first parameter for element function i. In addition, ISTEPA(n_e+1) is the position in EPVALU of the last parameter for element function n_e plus one. See Table 2.4 for an illustration.

ICALCF is a rank-one INTENT(IN) array argument of dimension lcalcf and type INTEGER(ip_) whose first ncalcf components gives the indices of the nonlinear element functions whose values or derivatives are to be evaluated.

ltypee, lstaev, lelvar, lntvar, lstadh, lstepa, lcalcf, lfuval, lxvalu and lepvlu are all scalar INTENT(IN) arguments of type INTEGER(ip_). They will have been set to the actual lengths of ITYPEE, ISTAEV, IELVAR, INTVAR, ISTADH, ISTEPA, ICALCF, FUVALS, XVALUE and EPVALU respectively.

Restrictions: ltypee $\geq$ prob%nel, lstaev $\geq$ prob%nel+1, lelvar $\geq$ ISTAEV(prob%nel+1)-1, lntvar $\geq$ prob%nel+1, lstadh $\geq$ prob%nel+1, lstepa $\geq$ prob%nel+1, lcalcf $\geq$ ncalcf, lfuval $\geq$ ISTADH(prob%nel+1)-1, lxvalu $\geq$ prob%n, and lepvlu $\geq$ ISTEPA(prob%nel+1)-1.

ifflag is a scalar INTENT(IN) argument of type INTEGER(ip_), whose value defines whether it is the values of the element functions that are required (ifflag = 1) or if it is the derivatives (ifflag > 1). Possible values and their requirements are:

ifflag = 1. The values of nonlinear element functions ICALCF(i), i =1,..., ncalcf, are to be computed and placed in FUVALS(ICALCF(i)).

ifflag = 2. The gradients of nonlinear element functions ICALCF(i), i =1,..., ncalcf, are to be computed. The gradient of the ICALCF(i)-th element is to be placed in the segment of FUVALS starting at INTVAR(ICALCF(i)) (see Table 2.8).

ifflag = 3. The gradients and Hessians of nonlinear element functions ICALCF(*i*), *i* = 1,..., ncalcf, are to be computed. The gradient of the ICALCF(*i*)-th element is to be placed in the segment of FUVALS starting at INTVAR(ICALCF(*i*)) (see Table 2.8). The Hessian of this element should be placed in the segment of FUVALS starting at ISTADH(ICALCF(*i*)) (see Table 2.9).

N.B. If the user intends to use approximate second derivatives (control%second_derivatives > 0, see Section 2.2.2), LANCELOT_solve will never call ELFUN with ifflag = 3, so the user need not provide Hessian values. Furthermore, if the user intends to use approximate first derivatives (control%first_derivatives > 0, see Section 2.2.2), LANCELOT_solve will never call ELFUN with ifflag = 2 or 3, so the user need then not provide gradient or Hessian values.

ifstat is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if all of the required values have been found, and to any nonzero value if, for any reason, one or more of the required values could not be determined. For instance, if the value of a nonlinear element (or its derivative) was required outside of its domain of definition, a nonzero value of ifstat should be returned.

2.4.5 Element function values and flexible derivatives via internal evaluation

If the argument ELFUN_flexible is present when calling LANCELOT_solve, the user is expected to provide a subroutine of that name to evaluate function or derivative values (with respect to internal variables) of (a given subset of) the element functions. **N.B.** This routine is only permitted if the argument ELDERS is also present. The routine has the following argument list:

```
SUBROUTINE ELFUN_flexible ( FUVALS, XVALUE, EPVALU, ncalcf, ITYPEE, ISTAEV, IELVAR, &
                            INTVAR, ISTADH, ISTEPA, ICALCF, ltypee, lstaev, lelvar, &
                            lntvar, lstadh, lstepa, lcalcf, lfuval, lxvalu, lepvlu, &
                            llders, ifflag, ELDERS, ifstat )
```

FUVALS, FUVALS, XVALUE, EPVALU, ncalcf, ITYPEE, ISTAEV, IELVAR, INTVAR, ISTADH, ISTEPA, ICALCF, ltypee, lstaev, lelvar, lntvar, lstadh, lstepa, lcalcf, lfuval, lxvalu, lepvlu and ifstat are all exactly as described for subroutine ELFUN in Section 2.4.4

llders is a scalar INTENT(IN) argument of type INTEGER(ip_), that will have been set to the actual length of ELDERS.

Restriction: llders ≥ prob%nel.

ifflag is a scalar INTENT(IN) argument of type INTEGER(ip_), whose value defines whether it is the values of the element functions that are required (ifflag = 1) or if it is the derivatives (ifflag > 1). If ifflag = 1, the values of nonlinear element functions ICALCF(*i*), *i* = 1,..., ncalcf, are to be computed and placed in FUVALS(ICALCF(*i*)). If ifflag > 1, those derivative values specified by ELDERS (see below) are required.

ELDERS is an rank-two INTENT(INOUT) array argument of shape (2, llders) and type INTEGER(ip_) that specifies what first- and second-derivative information (if any) is required for each nonlinear element function when ifflag > 1. Specifically, in this case, the gradient of each nonlinear element function ICALCF(*i*), *i* = 1,..., ncalcf, for which ELDERS(1,ICALCF(*i*)) ≤ 0 should be computed. The gradient of the ICALCF(*i*)-th element is to be placed in the segment of FUVALS starting at INTVAR(ICALCF(*i*)) (see Table 2.8). Furthermore, if additionally ELDERS(2,ICALCF(*i*)) ≤ 0, the Hessian of the nonlinear element function ICALCF(*i*) should be computed and placed in the segment of FUVALS starting at ISTADH(ICALCF(*i*)) (see Table 2.9).

2.4.6 Group function values and derivatives via internal evaluation

If the argument GROUP is present when calling LANCELOT_solve, the user is expected to provide a subroutine of that name to evaluate function or derivative values of (a given subset of) the group functions. The routine has the following argument list:

```
SUBROUTINE GROUP ( GVALUE, lgvalu, FVALUE, GPVALU, ncalcg, ITYPEG, ISTGPA, ICALCG, &
                   ltypeg, lstgpa, lcalcg, lfvalu, lgpvlu, derivs, igstat )
```

GVALUE is a rank-two INTENT(OUT) array argument of shape (lgvalu, 3) and type REAL(rp_). The value and first and second derivative of the i-th group function are held in GVALUE(i,1), GVALUE(i,2) and GVALUE(i,3) respectively.

lgvalu is a scalar INTENT(IN) argument of type INTEGER(ip_), that will have been set to the actual length of the leading dimension of GVALUE.

Restriction: lgvalu $\geq$ prob%ng.

FVALUE is a rank-one INTENT(IN) array argument of dimension lfvalu and type REAL(rp_) whose i-th component contains the value of group variable at which the subroutine is required to evaluate the value or derivatives of the i-th group function.

GPVALU is a rank-one INTENT(IN) array argument of dimension lgpvlu and type REAL(rp_) that contains the values of the group parameters $\mathbf{p}_i^g$, $i = 1,\ldots,n_g$. The indices for the parameters for group i immediately precede those for group $i+1$, and each group's parameters appear in a contiguous list. See Table 2.5 for an illustration.

ncalcg is a scalar INTENT(IN) argument of type INTEGER(ip_), that specifies how many of the group functions or their derivatives are to be evaluated.

ITYPEG is a rank-one INTENT(IN) array argument of dimension ltypeg and type INTEGER(ip_) whose i-th component specifies the type of group i.

ISTGPA is a rank-one INTENT(IN) array argument of dimension lstgpa and type INTEGER(ip_) whose i-th component gives the position in GPVALU of the first parameter for group function i. In addition, ISTGPA(n_g+1) is the position in GPVALU of the last parameter for element function n_g plus one. See Table 2.5 for an illustration.

ICALCG is a rank-one INTENT(IN) array argument of dimension lcalcg and type INTEGER(ip_) whose first ncalcg components gives the indices of the nonlinear group functions whose values or derivatives are to be evaluated.

ltypeg, lstgpa, lcalcg, lfvalu and lgpvlu are all scalar INTENT(IN) arguments of type INTEGER(ip_). They will have been set to the actual lengths of ITYPEG, ISTGPA, ICALCG, FVALUE and GPVALU.

Restrictions: ltypeg $\geq$ prob%n, lstgpa $\geq$ prob%ng+1, lcalcg $\geq$ ncalcg, lfvalu $\geq$ prob%ng, and lgpvlu $\geq$ ISTGPA(prob%ng+1)-1.

derivs is a scalar INTENT(IN) argument of type default LOGICAL. When derivs is .FALSE., the subroutine must return the values of group functions ICALCG(i), i =1,..., ncalcg in GVALUE(ICALCF(i),1). When derivs is .TRUE., the subroutine must return the first and second derivatives of group functions ICALCG(i), i =1,..., ncalcg in GVALUE(ICALCF(i),2) and GVALUE(ICALCF(i),3) respectively.

igstat is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if all of the required values have been found, and to any nonzero value if, for any reason, one or more of the required values could not be determined. For instance, if the value of a group function (or its derivative) was required outside of its domain of definition, a nonzero value of igstat should be returned.

2.4.7 Reverse Communication Information

When a return is made from LANCELOT_solve with inform%status set negative, LANCELOT_solve is asking the user for further information—this will happen if the user has chosen not to evaluate element or group values internally (see Section 2.4), or if the user wishes to provide his or her own preconditioner. The user should normally compute the required information and re-enter LANCELOT_solve with inform%status unchanged (but see inform%status $= -11$, below, for an exception).

Possible values of inform%status and the information required are

inform%status $= -1$. The user should compute the function and, if they are available, derivative values of the nonlinear element functions numbered ICALCF(*i*) for $i = 1, \ldots,$ ncalcf (see below).

inform%status $= -2$. The user should compute the function and derivative values of all non-trivial nonlinear group functions, numbered ICALCG(*i*) for $i = 1, \ldots,$ ncalcg (see below).

inform%status $= -3$. The user should compute the function values of the nonlinear element functions numbered ICALCF(*i*) for $i = 1, \ldots,$ ncalcf (see below).

inform%status $= -4$. The user should compute the function values of all non-trivial nonlinear group functions numbered ICALCG(*i*) for $i = 1, \ldots,$ ncalcg (see below).

inform%status $= -5$. If exact derivatives are available, the user should compute the derivative values of all nonlinear element functions numbered ICALCF(*i*) for $i = 1, \ldots,$ ncalcf. The user should also compute the derivative values of all non-trivial nonlinear group functions numbered ICALCG(*i*) for $i = 1, \ldots,$ ncalcg (see below).

inform%status $= -6$. The user should compute the derivative values of all nonlinear element functions numbered ICALCF(*i*) for $i = 1, \ldots,$ ncalcf (see below).

inform%status $= -7$. The user should compute the function values of the nonlinear element functions numbered ICALCF(*i*) for $i = 1, \ldots,$ ncalcf (see below).

inform%status $= -8, -9, -10$. The user should compute the product $\mathbf{q} = \mathbf{M}\mathbf{d}$ of a preconditioning matrix $\mathbf{M}$ with the vector $\mathbf{d}$ and return the value in $\mathbf{q}$ (see below).

When inform%status $= -1$, -3 or -7, the user must return the values of all of the nonlinear element functions $e_k(x_k^e)$, $k =$ICALCF(*i*), $i = 1, \ldots,$ ncalcf. The functions are to be evaluated at the point x given in the array XT. The k-th function value must be placed in FUVALS(k).

When inform%status $= -1, -5$ or -6 and all first derivatives are available (that is when control%first_derivatives = 0), the user must return the values of the gradients, *with respect to their internal variables*, of all the nonlinear element functions $e_k(\mathbf{x}_k^e, \mathbf{p}_k^e)$, $k =$ICALCF(*i*), $i = 1, \ldots,$ ncalcf—if a subset of the gradients are available (those for which ELDERS(1,*k*) $\leq$ 0), only these need be provided. The gradients are to be evaluated at the point $\mathbf{x}$ given in the array XT. The gradient with respect to internal variable i of the k-th nonlinear element,

$$\frac{\partial e_k}{\partial u_i},$$

must be placed in FUVALS(INTVAR(*k*)+*i*−1). If, in addition, exact second derivatives are to be provided (control%second_derivatives = 0), the user must return the values of the Hessian matrices, *with respect to their internal variables*, of the same nonlinear element functions evaluated at $\mathbf{x}$—again, if a subset of the Hessians are available (those for which ELDERS(2,*k*) $\leq$ 0), only these need be provided. Only the "upper triangular" part of the required Hessians should be specified. The component of the Hessian of the k-th nonlinear element with respect to internal variables i and j, $i \leq j$,

$$\frac{\partial^2 e_k}{\partial u_i \partial u_j},$$

must be placed in FUVALS(ISTADH(*k*)+$(j(j-1)/2)+i-1$).

When inform%status $= -2$ or -4, the user must return the values of all the group functions g_k, $k =$ICALCG(*i*), $i = 1, \ldots,$ ncalcg. The k-th such function should be evaluated with the argument FT(*k*) and the result placed in GVALS(*k*,1).

When inform%status $= -2$ or -5, the user must return the values of the first and second derivatives of each of the group functions g_k, $k =$ICALCG(*i*), $i = 1, \ldots,$ ncalcg, with respect to its argument. The derivatives of the k-th such function should be evaluated with the argument FT(*k*). The first derivative of the k-th group function should be placed in GVALS(*k*,2) and the corresponding second derivative returned in GVALS(*k*,3).

When inform%status $= -8, -9$ or -10, the user must return the values of the components Q(IVAR(i)) $= q_i$, $i = 1, \ldots,$ inform%nvar, such that $\mathbf{q} = \mathbf{M}\mathbf{d}$ and where $d_i =$DGRAD(i) $i = 1, \ldots,$ inform%nvar. Here $\mathbf{M}$ is a symmetric positive definite approximation to the inverse of the matrix whose i-th row and column are the IVAR(i)-th row and column of the Hessian matrix of augmented Lagrangian function (see section 4), $i = 1, \ldots,$ inform%nvar. These values can only occur if control%linear_solver = 3.

If the user does not wish, or is unable, to compute an element or group function at a particular argument returned from LANCELOT_solve, inform%status may be reset to -11 and LANCELOT_solve re-entered. LANCELOT_solve will treat such a re-entry as if the current iteration had been unsuccessful and reduce the trust-region radius. This facility is useful when, for instance, the user is asked to evaluate a function at a point outside its domain of definition.

2.5 Warning and error messages

If inform%status is positive on return from LANCELOT_solve, an error has been detected. The user should correct the error and restart the minimization. Possible values of inform%status and their consequences are:

inform%status $= 1$. More than control%maxit iterations have been performed. This is often a symptom of incorrectly programmed derivatives or of the preconditioner used being insufficiently effective. Recheck the derivatives. Otherwise, increase control%maxit and re-enter LANCELOT_solve at the best point found so far.

inform%status $= 2$. The trust-region radius has become too small. This is often a symptom of incorrectly programmed derivatives or of requesting more accuracy in the projected gradient than is reasonable on the user's machine. If the projected gradient is small, the minimization has probably succeeded. Otherwise, recheck the derivatives.

inform%status $= 3$. The step taken during the current iteration is so small that no difference will be observed in the function values. This sometimes occurs when too much accuracy is required of the final gradient. If the projected gradient is small, the minimization has probably succeeded.

inform%status $= 4$. One of the INTEGER arrays has been initialized with insufficient space. A message indicating which array is at fault and the required space will be printed on unit number control%error.

inform%status $= 5$. One of the REAL (DOUBLE PRECISION in the D version) arrays has been initialized with insufficient space. A message indicating which array is at fault and the required space will be printed on unit number control%error.

inform%status $= 6$. One of the LOGICAL arrays has been initialized with insufficient space. A message indicating which array is at fault and the required space will be printed on unit number control%error.

inform%status $= 7$. One or more of the components of the array KNDOFG does not have the value 0, 1 or 2.

inform%status $= 8$. The problem does not appear to have a feasible solution. Check the constraints and try starting with different initial values for $\mathbf{x}$.

inform%status $= 9$. One of the arrays prob%KNDOFG, prob%C or prob%Y is either not associated or is not large enough. (Re-)allocate the arrays to have dimensions at least prob%ng.

inform%status $= 10$. The user has provided ELFUN to perform internal element evaluation, but one or both of prob%ISTEPA and prob%EPVALU is not associated or at least one is not large enough. (Re-)allocate (and reset) prob%ISTEPA to have dimension at least prob%nel+1 and prob%EPVALU to have dimension at least prob%ISTEPA(prob%nel+1)-1.

inform%status $= 11$. The user has provided GROUP to perform internal group evaluation, but one or both of prob%ISTGPA and prob%GPVALU is not associated or at least one is not large enough. (Re-)allocate (and reset) prob%ISTGPA to have dimension at least prob%ng+1 and prob%GPVALU to have dimension at least prob%ISTGPA(prob%ng+1)-1.

inform%status = 12. An internal array allocation or de-allocation has failed. The name of the offending array is given in inform%bad_alloc, and the allocation status is inform%alloc_status.

inform%status = 13. One or more of the problem functions cannot be computed at the initial point. Try starting with a different initial value for **x**.

inform%status = 14. The user has forced termination of LANCELOT_solve by removing the file named control%alive_file from unit unit control%alive_unit.

inform%status = 15. One or more of the problem dimensions prob%n, prob%ng or prob%nel is too small. Ensure that prob%n $>$ 0, prob%ng $>$ 0 or prob%nel $\geq$ 0.

inform%status = 16. Both optional arguments ELDERS and ELFUN are present at the same time. Either remove ELDERS (or ELFUN) or replace ELFUN by ELFUN_flexible.

inform%status = 17. The optional argument ELFUN_flexible is present, but ELDERS is absent. Either include ELDERS or remove ELFUN_flexible.

inform%status = 18. The current value of the merit function (1.3), inform%aug, is smaller than its smallest permitted value, control%min_aug. Check the formulation to see if this seems reasonable.

inform%status = 19. The maximum CPU time limit has been exceed.

inform%status = 26. A requested linear solver is unavailable.

2.6 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type LANCELOT B_control_type (see Section 2.2.2), by reading an appropriate data specification file using the subroutine LANCELOT B_read_specfile. This facility is useful as it allows a user to change LANCELOT B control parameters without editing and recompiling programs that call LANCELOT B.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by LANCELOT B_read_specfile must start with a "BEGIN LANCELOT B" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by LANCELOT_read_specfile .. )
  BEGIN LANCELOT
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by LANCELOT_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN LANCELOT B" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN LANCELOT SPECIFICATION
```

and

```
END LANCELOT SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN LANCELOT B" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of five different types, namely integer, logical, real, character string, or symbolic. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively), while character strings are arbitrary collections of default Fortran character constants (without quotation marks). A symbolic value is a special string (defined in the GALAHAD_SYMBOLS module), that may help to express a control parameter for LANCELOT B in a "language" that is close to natural. Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when LANCELOT B_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by LANCELOT B_read_specfile.

2.6.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL LANCELOT_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type LANCELOT B_control_type (see Section 2.2.2). Default values should have already have set, perhaps by calling LANCELOT B_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.2.2) of control that each affects are given in Table 2.10; permitted string values are described in Table 2.11.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

2.7 Information printed

The user is able to control the amount of intermediate printing performed in the course of the minimization. Printing is under the control of the parameter control%print_level and output is sent to I/O unit number control%error. Possible values of control%print_level and the levels of output produced are as follows.

control%print_level ≤ 0. No printing, except warning messages, will be performed.

control%print_level ≥ 1. Details of the minimization function will be output. This includes the number of variables, groups and nonlinear elements which are used and a list of the variables which occur in each of the linear and nonlinear elements in every group.

If the current iterate provides an acceptable estimate of the minimizer of the augmented Lagrangian function, the two-norm of the general constraints and the current value of the penalty parameter are given.

control%print_level $= 1$. A simple one line description of each iteration is given. This includes the iteration number, the number of derivative evaluations that have been made, the number of conjugate-gradient iterations that

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
alive-device	%alive_unit	integer
print-level	%print_level	integer
maximum-number-of-iterations	%maxit	integer
start-print	%start_print	integer
stop-print	%stop_print	integer
iterations-between-printing	%print_gap	integer
linear-solver-used	%linear_solver	string
number-of-lin-more-vectors-used	%icfact	integer
semi-bandwidth-for-band-preconditioner	%semibandwidth	integer
maximum-dimension-of-schur-complement	%max_sc	integer
unit-number-for-temporary-io	%io_buffer	integer
more-toraldo-search-length	%more_toraldo	integer
history-length-for-non-monotone-descent	%non_monotone	integer
first-derivative-approximations	%first_derivatives	string
second-derivative-approximations	%second_derivatives	string
primal-accuracy-required	%stopc	real
dual-accuracy-required	%stopg	real
minimum-merit-value	%min_aug	real
inner-iteration-relative-accuracy-required	%acccg	real
initial-trust-region-radius	%initial_radius	real
maximum-radius	%maximum_radius	real
eta-successful	%eta_successful	real
eta-very-successful	%eta_very_successful	real
eta-extremely-successful	%eta_extremely_successful	real
gamma-smallest	%gamma_smallest	real
gamma-decrease	%gamma_decrease	real
gamma-increase	%gamma_increase	real
mu-meaningful-model	%mu_meaningful_model	real
mu-meaningful-group	%mu_meaningful_group	real
initial-penalty-parameter	%initial_mu	real
no-dual-updates-until-penalty-parameter-below	%mu_tol	real
initial-primal-accuracy-required	%firstc	real
initial-dual-accuracy-required	%firstg	real
pivot-tolerance-used	%SILS_cntl%u	real
maximum-cpu-time-limit	cpu_time_limit	real
quadratic-problem	%quadratic_problem	logical
two-norm-trust-region-used	%two_norm_tr	logical
exact-GCP-used	%exact_gcp	logical
magical-steps-allowed	%magical_steps	logical
subproblem-solved-accurately	%accurate_bqp	logical
structured-trust-region-used	%structured_tr	logical
print-for-maximization	%print_max	logical
print-full-solution	%full_solution	logical
alive-filename	%alive_file	character

Table 2.10: Specfile commands and associated components of control. See Table 2.11 for permitted string values.

command	string value	control component assignment
linear-solver-used	CG	%linear_solver = 1
	DIAGONAL_CG	%linear_solver = 2
	USERS_CG	%linear_solver = 3
	EXPANDING_BAND_CG	%linear_solver = 4
	MUNKSGAARD_CG	%linear_solver = 5
	SCHNABEL_ESKOW_CG	%linear_solver = 6
	GMPS_CG	%linear_solver = 7
	BAND_CG	%linear_solver = 8
	LIN_MORE_CG	%linear_solver = 9
	MULTIFRONTAL	%linear_solver = 11
	MODIFIED_MULTIFRONTAL	%linear_solver = 12
first-derivative-approximations	EXACT	%first_derivatives = 0
	FORWARD	%first_derivatives = 1
	CENTRAL	%first_derivatives = 2
second-derivative-approximations	EXACT	%second_derivatives = 0
	BFGS	%second_derivatives = 1
	DFP	%second_derivatives = 2
	PSB	%second_derivatives = 3
	SR1	%second_derivatives = 4

Table 2.11: String values and their interpretation.

have been performed, the current value of the augmented Lagrangian function, the (two-) norm of the projected gradient, the ratio ρ of the actual to predicted decrease in augmented Lagrangian function value achieved, the current trust-region radius, the norm of the step taken, an indication of how the direct or iterative method ended, the number of variables which lie away from their bounds and the total time spent on the minimization.

control%print_level $= 2$. In addition to the information output with control%print_level = 1, a short description of the approximate solution to the inner-iteration linear system is given. Before a successful (inform%status = 0) exit, details of the estimate of the minimizer and the gradient of the augmented Lagrangian function are given.

control%print_level $= 3$. A list of the current iteration number, the value of the augmented Lagrangian function, the number of derivative evaluations that have been made, the (two-) norm of the projected gradient, the number of conjugate gradients iterations that have been performed and the current trust-region radius are given, followed by the current estimate of the minimizer. The values of the reduction in the model of the augmented Lagrangian function and the actual reduction in this function, together with their ratio, are also given. Before a successful (inform%status = 0) exit, details of the estimate of the minimizer and the gradient of the augmented Lagrangian function are given.

If the current iterate also provides an acceptable estimate of the minimizer of the augmented Lagrangian function, values of the general constraints and estimates of the Lagrange multipliers are also given.

control%print_level $= 4$. In addition to the information output with control%print_level= 3, the gradient of the augmented Lagrangian function at the current estimate of the minimizer is given. Full details of the approximate solution to the inner-iteration linear system are also given. This level of output is intended as a debugging aid for the expert only.

control%print_level $= 5$. In addition to the information output with control%print_level = 4, the diagonal elements of the second derivative approximation are given.

control%print_level ≥ 6. In addition to the information output with control%print_level = 5, the second derivative approximations (taken with respect to the internal variables) to each nonlinear element function are given.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: The user must provide an external RANGE subroutine (see Section 2.4.3), and optionally may provide external ELFUN (see Section 2.4.4), ELFUN_flexible (see Section 2.4.5) and GROUP (see Section 2.4.6) subroutines. Since these routines are all dummy arguments to LANCELOT_solve their names may be adapted for the user's purposes.

Other modules used directly: GALAHAD_CPU_time, GALAHAD_SPECFILE, LANCELOT_INITW, LANCELOT_OTHERS, LANCELOT_HSPRD, LANCELOT_CAUCHY, LANCELOT_CG, LANCELOT_PRECN, LANCELOT_FRNTL, LANCELOT_STRUTR, GALAHAD_SMT, GALAHAD_SILS, GALAHAD_SCU, LANCELOT_ASMBL, LANCELOT_EXTEND.

Input/output: No input; output on device numbers control%out and control%error. Output is provided under the control of control%print_level. If the user supplies positive unit numbers to control%out and control%error, messages are printed to the user supplied units. However if one of unit numbers is non-positive, printing to this unit is suppressed.

Restrictions: prob%n > 0, prob%ng > 0, prob%nel ≥ 0.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

The basic method implemented within LANCELOT B is described in detail by Conn, Gould and Toint (1991). The method used to solve the inner iteration subproblem is described by Conn, Gould and Toint (1988b). The concept of partial separability was first suggested by Griewank and Toint (1982). The extension to group partially separable functions was given by Conn, Gould and Toint (1990). Also see Chapter 3 of the LANCELOT A manual.

The Lagrangian function associated with objective function (1.1) and general constraints (1.2) is the composite function

$$\ell(\mathbf{x},\mathbf{y}) = f(\mathbf{x}) + \sum_{i \in \mathcal{G}_C} y_i c_i(\mathbf{x}).$$

The scalars y_i are known as Lagrange multiplier estimates. At a solution $\mathbf{x}^\star$ to the constrained minimization problem, there are Lagrange multipliers $\mathbf{y}^\star$ for which the components of the gradient of the Lagrangian function $\partial \ell(\mathbf{x}^\star,\mathbf{y}^\star,)/\partial x_i = 0$ whenever the corresponding variable $x_i^\star$ lies strictly between its lower and upper bounds. It is useful, but not essential, for the initial Lagrange multiplier estimates to be close to $\mathbf{y}^\star$.

The augmented Lagrangian function is the composite function

$$\phi(\mathbf{x},\mathbf{y},\mu) = \ell(\mathbf{x},\mathbf{y}) + \frac{1}{2\mu} \sum_{i \in \mathcal{G}_C} (c_i(\mathbf{x}))^2, \tag{4.1}$$

where μ is known as the penalty parameter. An inner iteration is used to find an approximate minimizer of (4.1) within the feasible box for fixed values of the penalty parameter and Lagrange multiplier estimates. The outer iteration of LANCELOT B automatically adjusts the penalty parameter and Lagrange multiplier estimates to ensure convergence of

these approximate minimizers to a solution of the constrained optimization problem. Each inner iteration is terminated as soon as a norm of the projected gradient,

$$\min(\max(\mathbf{x} - \nabla_x \phi(\mathbf{x}, \mathbf{y}, \mu), \mathbf{x}^l), \mathbf{x}^u) - \mathbf{x},$$

is sufficiently small.

In the inner iteration, a step from the current estimate of the solution is determined using a trust-region approach. That is, a quadratic model of the augmented Lagrangian function is approximately minimized within the intersection of the constraint "box" and another convex region, the trust-region. This minimization is carried out in two stages. Firstly, the so-called generalized Cauchy point for the quadratic subproblem is found. (The purpose of this point is to ensure that the algorithm converges and that the set of bounds which are satisfied as equations at the solution is rapidly identified.) Thereafter an improvement to the quadratic model is sought using either a direct-matrix or truncated conjugate-gradient algorithm. The trust-region size is increased if the reduction obtained in the objective function is reasonable when compared with the reduction predicted by the model and reduced otherwise.

A central idea is that a collection of small matrices approximating the Hessian matrices of each e_k is used and updated at every iteration using one of a number of possible updating formulae. Augmented Lagrangian function values and derivatives are assembled from these components as required.

The strategy for treating bound constraints is based on the usual projection device and is described in detail in Conn, Gould and Toint (1988a). Of the new features in LANCELOT B, structured trust regions are described in Conn, Gould and Toint (2000), Section 10.2, and non-monotone descent, *ibid*, Section 10.1, while the alternative projection is due to Moré and Toraldo (1991), and the alternative incomplete Cholesky factorization is described by Lin and Moré (1998)

References:

The basic method is described in detail in

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1992). LANCELOT. A fortran package for large-scale nonlinear optimization (release A). Springer Verlag Series in Computational Mathematics 17, Berlin,

and details of its computational performance may be found in

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1996). Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization Mathematical Programming **73** 73-110.

Convergence properties of the method are described in

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1991). A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds. SIAM Journal on Numerical Analysis **28** 545-572,

and

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1988a). Global convergence of a class of trust region algorithms for optimization with simple bounds. SIAM Journal on Numerical Analysis **25** 433-460,

while details of the inner iteration are provided by

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1988b). Testing a class of methods for solving minimization problems with simple bounds on the variables. Mathematics of Computation **50** 399-430.

Partial separability was introduced by

A. Griewank and Ph. L. Toint (1982). Partitioned variable metric updates for large structured optimization problems. Numerische Mathematik **39** 119-137,

and its generalization to group partial separability was given by

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1990). An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. In "Computing Methods in Applied Sciences and Engineering" (R. Glowinski and A. Lichnewsky, eds), SIAM, Philadelphia, 42-51.

Many of the newer issues are discussed by

A. R. Conn, N. I. M. Gould and Ph. L. Toint (2000). Trust Region Methods. SIAM, Philadelphia,

while the alternative incomplete Cholesky factorization is due to

C.-J. Lin and J. J. Moré (1999). Incomplete Cholesky factorizations with limited memory. SIAM Journal on Scientific Computing **21** 21-45,

and the alternative projection originally described in

J. J. Moré and G. Toraldo (1991) On the solution of large quadratic programming problems with bound constraints. SIAM Journal on Optimization **1** 93-113,

and given in the context used here by

C.-J. Lin and J. J. Moré (1999b). Newton's method for large bound-constrained optimization problems, SIAM Journal on Optimization **9** 1100-1127.

5 EXAMPLE OF USE

We now consider the small example problem,

$$\text{minimize } f(x_1,x_2,x_3) = x_1^2 + x_2\sin(x_1+x_3) + 3x_2^4x_3^4 + x_2 + 2x_1^2x_2^2$$

subject to the general constraint

$$c(x_1,x_2,x_3) = \cos(x_1+2x_2-1) = 0$$

and the bounds $-1 \leq x_2 \leq 1$ and $1 \leq x_3 \leq 2$. There are a number of ways of casting this problem in the framework required by `LANCELOT_solve`. Here, we consider partitioning f into groups as

$$\begin{array}{ccccccccccc} (x_1)^2 & + & (x_2\sin(x_1+x_3)) & + & 3(x_2x_3)^4 & + & (x_2) & + & 2(x_1x_2)^2 \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ \text{group 1} & & \text{group 2} & & \text{group 3} & & \text{group 4} & & \text{group 5} \end{array}$$

Similarly, we write c as

$$\begin{array}{c} \cos(x_1+2x_2-1). \\ \uparrow \\ \text{group 6} \end{array}$$

Notice the following:

1. Group 1 uses the non-trivial group function $g_1(\alpha) = \alpha^2$. The group contains a single *linear* element; the element function is x_1.

2. Group 2 uses the trivial group function $g_2(\alpha) = \alpha$. The group contains a single *nonlinear* element; this element function is $e_1(x_1,x_2,x_3) = x_2\sin(x_1+x_3)$. The element function has *three* elemental variables, v_1, v_2 and v_3, say, (with $v_1 = x_2$, $v_2 = x_1$ and $v_3 = x_3$), but may be expressed in terms of *two* internal variables u_1 and u_2, say, where $u_1 = v_1$ and $u_2 = v_2 + v_3$.

3. Group 3 uses the non-trivial group function $g_3(\alpha) = \alpha^4$, weighted by the factor 3. Again, the group contains a single *nonlinear* element; this element function is $e_2(x_2,x_3) = x_2x_3$. The element function has *two* elemental variables, v_1 and v_2, say, (with $v_1 = x_2$ and $v_2 = x_3$). This time, however, there is no useful transformation to internal variables.

4. Group 4 again uses the trivial group function $g_4(\alpha) = \alpha$. This time the group contains a single *linear* element x_2.

5. Group 5 uses the group function $g_5(\alpha) = \alpha^2$, weighted by the factor 2. Notice that this is the same *type* as occurred in group 1. The group contains a single nonlinear element, $e_3(x_1,x_2) = x_1x_2$. But notice that this element is of the same *type* (i.e., v_1v_2 for appropriate v_1 and v_2) as e_2 featured in group 3 although it uses different elemental variables (this time $v_1 = x_1$ and $v_2 = x_2$).

6. Finally, group 6 uses the non-trivial group function $g_6(\alpha) = \cos(\alpha)$. The group contains a single linear element; the element function is $x_1 + 2x_2 - 1$.

Thus we see that we can consider our objective and constraint functions to be made up of six group functions; the first, third, fifth and sixth are non-trivial so we need to provide function and derivative values for these when prompted by `LANCELOT_solve`. Of the four non-trivial groups, there are only three different types, since groups one and five are structurally the same. There are three nonlinear elements, one each from groups two, three and five. Again this means that we need to provide function and derivative values for these when required by `LANCELOT_solve`, but there is a slight simplification as elements two and three are of the same type. Finally, one of these elements, the first, has a useful transformation from elemental to internal variables so the routine `RANGE` must be set to provide this transformation.

The problem involves three variables, six groups and three nonlinear elements so we set `prob%n` = 3, `prob%ng` = 6 and `prob%nel` = 3. As the first, third, fifth and sixth group functions are non-trivial, we set `prob%GXEQX` as:

```
          i            1        2        3        4        5        6
prob%GXEQX(i)    .FALSE.   .TRUE.   .FALSE.   .TRUE.   .FALSE.   .FALSE.
```

Non-trivial groups 1 and 5 involve functions of the same type (we label this a "type 1" group; the actual label is unimportant, so long as groups of different types have different labels), and these are different from those for groups 2 and 6 (which we call "types 2" and "3")—by convention, trivial groups are of "type 0" . So we set `prob%ITYPEG` as:

```
          i          1  2  3  4  5  6
prob%ITYPEG(i)       1  0  2  0  1  3
```

The first five groups are associated with the objective function while the sixth defines a constraint so we set `prob%KNDOFG` as:

```
          i          1  2  3  4  5  6
prob%KNDOFG(i)       1  1  1  1  1  2
```

The first nonlinear element occurs in group 2, the second in group 3 and the last in group 5; each element is unweighted. We thus set `prob%IELING`, `prob%ESCALE` and `prob%ISTADG` as:

```
          i          1    2    3
prob%IELING(i)       1    2    3
prob%ESCALE(i)     1.0  1.0  1.0

          i          1  2  3  4  5  6  7
prob%ISTADG(i)       1  1  2  3  3  4  4
```

The first nonlinear element function is not a quadratic function of its internal variables so we set `control%quadratic_problem=.FALSE.`. The second and third nonlinear element functions are of the same type (we number this a "type 2" element), while the first nonlinear element is different (a "type 1" element). So we set `prob%ITYPEE` as:

```
          i          1  2  3
prob%ITYPEE(i)       1  2  2
```

Nonlinear element one, e_1 assigns variables x_2, x_1 and x_3 to its elemental variables, while the second and third nonlinear elements e_2 and e_3 assign variables x_2 and x_3, and x_1 and x_2 to their elemental variables, respectively. Hence `prob%IELVAR` contains:

i	1	2	3	4	5	6	7
prob%IELVAR(i)	2	1	3	2	3	1	2
	⟵	e_1	⟶	←	e_2	→	← e_3 →

In this vector, we now locate the position of the first variable of each nonlinear element, and build the vector prob%ISTAEV as follows:

i	1	2	3	4
prob%ISTAEV(i)	1	4	6	8

All three nonlinear elements have two internal variables — elements two and three using their elemental variables as internals — so we set prob%INTVAR as follows:

i	1	2	3	4
prob%INTVAR(i)	2	2	2	-

As the first nonlinear element has a useful transformation between elemental and internal variables, while the other two do not, prob%INTREP contains:

i	1	2	3
prob%INTREP(i)	.TRUE.	.FALSE.	.FALSE.

Turning to the linear elements, groups one, four and six each have a linear element. Those from groups one and four involve a single variable — x_1 for group one and x_2 for group four. That from group six involves the variables x_1 and x_3. All the coefficients, excepting the last, are one; the last is two. We thus set prob%ISTADA as:

i	1	2	3	4	5	6	7
prob%ISTADA(i)	1	2	2	2	3	3	5

and prob%ICNA and prob%A as

i	1	2	3	4
prob%ICNA(i)	1	2	1	2
prob%A(i)	1.0D+0	1.0D+0	1.0D+0	2.0D+0

Only the last group has a constant term so prob%B is set to:

i	1	2	3	4	5	6
prob%B(i)	0.0D+0	0.0D+0	0.0D+0	0.0D+0	0.0D+0	1.0D+0

The bounds for the problem are set in prob%BL and prob%BU. As the first variable is allowed to take any value, we specify lower and upper bounds of $\pm 10^{20}$. Thus we set

i	1	2	3
prob%BL(i)	-1.0D+20	-1.0D+0	-1.0D+0
prob%BU(i)	1.0D+20	1.0D+0	2.0D+0

The 3rd group is scaled by 3.0, the fifth by 2.0, while the others are unscaled. Thus we set prob%GSCALE to:

i	1	2	3	4	5	6
prob%GSCALE(i)	1.0D+0	1.0D+0	3.0D+0	1.0D+0	2.0D+0	1.0D+0

It is unclear that the variables are badly scaled so we set prob%VSCALE to:

```
                i            1        2        3
prob%VSCALE(i)   1.0D+0   1.0D+0   1.0D+0
```

Finally, we choose to allocate the names `Obj 1` to `Obj 5` to the four groups associated with the objective function and call the constraint group `Constraint`. The variables are called `x 1` to `x 3`. We thus set `prob%GNAMES` and `prob%VNAMES` to:

```
                i            1       2       3       4       5          6
prob%GNAMES(i)   Obj 1   Obj 2   Obj 3   Obj 4   Obj 5   Constraint
```

and

```
                i           1     2     3
prob%VNAMES(i)   x 1   x 2   x 3
```

Notice that, in our example, neither the group nor nonlinear element functions depends upon parameters, so both `prob%EPVALU` and `prob%GPVALU` are empty, and consequently all components of the parameter starting address arrays `prob%ISTEPA` and `prob%ISTGPA` should be set to `0`.

`LANCELOT_solve` leaves the computation of function and derivative values of the group and nonlinear element functions to the user—this may either be done internally (by providing suitable `GROUP` and `ELFUN` subroutines, or externally via reverse communication. We need only to specify the group functions for the non-trivial groups, groups 1, 3 and 5. Also note that the function and gradient values are only evaluated if the component is specified in `ICALCG`. A suitable `GROUP` subroutine might be the following:

```
SUBROUTINE GROUP ( GVALUE, lgvalu, FVALUE, GPVALU, ncalcg, &
                   ITYPEG, ISTGPA, ICALCG, ltypeg, lstgpa, &
                   lcalcg, lfvalu, lgpvlu, derivs, igstat )
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
INTEGER, INTENT( IN ) :: lgvalu, ncalcg
INTEGER, INTENT( IN ) :: ltypeg, lstgpa, lcalcg, lfvalu, lgpvlu
INTEGER, INTENT( OUT ) :: igstat
LOGICAL, INTENT( IN ) :: derivs
INTEGER, INTENT( IN ), DIMENSION ( ltypeg ) :: ITYPEG
INTEGER, INTENT( IN ), DIMENSION ( lstgpa ) :: ISTGPA
INTEGER, INTENT( IN ), DIMENSION ( lcalcg ) :: ICALCG
REAL ( KIND = wp ), INTENT( IN ), DIMENSION ( lfvalu ) :: FVALUE
REAL ( KIND = wp ), INTENT( IN ), DIMENSION ( lgpvlu ) :: GPVALU
REAL ( KIND = wp ), INTENT( OUT ), DIMENSION ( lgvalu, 3 ) :: GVALUE
INTEGER :: igrtyp, igroup, ipstrt, jcalcg
REAL ( KIND = wp ) :: alpha, alpha2
igstat = 0
DO jcalcg = 1, ncalcg
  igroup = ICALCG( jcalcg )
  igrtyp = ITYPEG( igroup )
  IF ( igrtyp == 0 ) CYCLE ! skip if the group is trivial
  ipstrt = ISTGPA( igroup ) - 1
  SELECT CASE ( igrtyp )
  CASE ( 1 )
    alpha = FVALUE( igroup )
    IF ( .NOT. derivs ) THEN
      GVALUE( igroup, 1 ) = alpha * alpha
    ELSE
      GVALUE( igroup, 2 ) = 2.0_wp * alpha
      GVALUE( igroup, 3 ) = 2.0_wp
    END IF
  CASE ( 2 )
    alpha  = FVALUE( igroup )
    alpha2 = alpha * alpha
```

```
      IF ( .NOT. derivs ) THEN
        GVALUE( igroup, 1 ) = alpha2 * alpha2
      ELSE
        GVALUE( igroup, 2 ) = 4.0_wp * alpha2 * alpha
        GVALUE( igroup, 3 ) = 12.0_wp * alpha2
      END IF
    CASE ( 3 )
      alpha  = FVALUE( igroup )
      IF ( .NOT. derivs ) THEN
        GVALUE( igroup, 1 ) = COS( alpha )
      ELSE
        GVALUE( igroup, 2 ) = - SIN( alpha )
        GVALUE( igroup, 3 ) = - COS( alpha )
      END IF
    END SELECT
  END DO
  RETURN
  END SUBROUTINE GROUP
```

Here, the logical parameter `derivs` set `.TRUE.` specifies that the first and second derivatives are required; a `.FALSE.` value will return the function values.

To evaluate the values and derivatives of the nonlinear element functions, the following `ELFUN` subroutine would be suitable:

```
SUBROUTINE ELFUN ( FUVALS, XVALUE, EPVALU, ncalcf, ITYPEE, ISTAEV, &
                   IELVAR, INTVAR, ISTADH, ISTEPA, ICALCF, ltypee, &
                   lstaev, lelvar, lntvar, lstadh, lstepa, lcalcf, &
                   lfuval, lxvalu, lepvlu, ifflag, ifstat )
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
INTEGER, INTENT( IN ) :: ncalcf, ifflag, ltypee, lstaev, lelvar, lntvar
INTEGER, INTENT( IN ) :: lstadh, lstepa, lcalcf, lfuval, lxvalu, lepvlu
INTEGER, INTENT( OUT ) :: ifstat
INTEGER, INTENT( IN ) :: ITYPEE( LTYPEE ), ISTAEV( LSTAEV ), IELVAR( LELVAR )
INTEGER, INTENT( IN ) :: INTVAR( LNTVAR ), ISTADH( LSTADH ), ISTEPA( LSTEPA )
INTEGER, INTENT( IN ) :: ICALCF( LCALCF )
REAL ( KIND = wp ), INTENT( IN ) :: XVALUE( LXVALU )
REAL ( KIND = wp ), INTENT( IN ) :: EPVALU( LEPVLU )
REAL ( KIND = wp ), INTENT( INOUT ) :: FUVALS( LFUVAL )
INTEGER :: ielemn, ieltyp, ihstrt, ilstrt, igstrt, ipstrt, jcalcf
REAL ( KIND = wp ) :: v1, v2, u1, u2, u3, cs, sn
ifstat = 0
DO jcalcf = 1, ncalcf
  ielemn = ICALCF( jcalcf )
  ilstrt = ISTAEV( ielemn ) - 1
  igstrt = INTVAR( ielemn ) - 1
  ipstrt = ISTEPA( ielemn ) - 1
  IF ( ifflag == 3 ) ihstrt = ISTADH( ielemn ) - 1
  ieltyp = ITYPEE( ielemn )
  SELECT CASE ( ieltyp )
  CASE ( 1 )
    u1 = XVALUE( IELVAR( ilstrt + 1 ) )
    u2 = XVALUE( IELVAR( ilstrt + 2 ) )
    u3 = XVALUE( IELVAR( ilstrt + 3 ) )
    v1 = u1
    v2 = u2 + u3
    cs = COS( v2 )
    sn = SIN( v2 )
    IF ( ifflag == 1 ) THEN
      FUVALS( ielemn ) = v1 * sn
    ELSE
      FUVALS( igstrt + 1 ) = sn
      FUVALS( igstrt + 2 ) = v1 * cs
```

```
        IF ( ifflag == 3 ) THEN
          FUVALS( ihstrt + 1 ) = 0.0_wp
          FUVALS( ihstrt + 2 ) = cs
          FUVALS( ihstrt + 3 ) = - v1 * sn
        END IF
      END IF
    CASE ( 2 )
      u1 = XVALUE( IELVAR( ilstrt + 1 ) )
      u2 = XVALUE( IELVAR( ilstrt + 2 ) )
      IF ( ifflag == 1 ) THEN
        FUVALS( ielemn ) = u1 * u2
      ELSE
        FUVALS( igstrt + 1 ) = u2
        FUVALS( igstrt + 2 ) = u1
        IF ( ifflag == 3 ) THEN
          FUVALS( ihstrt + 1 ) = 0.0_wp
          FUVALS( ihstrt + 2 ) = 1.0_wp
          FUVALS( ihstrt + 3 ) = 0.0_wp
        END IF
      END IF
    END SELECT
  END DO
  RETURN
  END SUBROUTINE ELFUN
```

The integer parameter ifflag specifies whether function values (`ifflag = 1`) or derivatives (`ifflag > 1`) are required. When `ifflag = 2`, just first derivatives are required, while `ifflag = 3` also requests second derivatives. Notice that the derivatives are taken with respect to the internal variables. For the first nonlinear element function, this means that we must transform from elemental to internal variables before evaluating the derivatives. Thus, as this function may be written as $f(v_1,v_2) = v_1 \sin v_2$, where $v_1 = u_1$ and $v_2 = u_2 + u_3$, the gradient and Hessian matrix are

$$\begin{pmatrix} \sin v_2 \\ v_1 \cos v_2 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & \cos v_2 \\ \cos v_2 & -v_1 \sin v_2 \end{pmatrix},$$

respectively. Notice that it is easy to specify the second derivatives for this example so we do so. Also note that the function and gradient values are only evaluated if the component is specified in `ICALCF`.

We must also specify the routine `RANGE` for our example. As we have observed, only the first element has a useful transformation. The transformation matrix is

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

(see Section 2.4.3). As a consequence, the following routine `RANGE` is appropriate:

```
SUBROUTINE RANGE( ielemn, transp, W1, W2, nelvar, ninvar, ieltyp, lw1, lw2 )
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
INTEGER, INTENT( IN ) :: ielemn, nelvar, ninvar, ieltyp, lw1, lw2
LOGICAL, INTENT( IN ) :: transp
REAL ( KIND = wp ), INTENT( IN ), DIMENSION ( lw1 ) :: W1
REAL ( KIND = wp ), INTENT( OUT ), DIMENSION ( lw2 ) :: W2
SELECT CASE ( ieltyp )
CASE ( 1 ) ! Element type 1 has a non-trivial transformation
  IF ( transp ) THEN
    W2( 1 ) = W1( 1 )
    W2( 2 ) = W1( 2 )
    W2( 3 ) = W1( 2 )
  ELSE
    W2( 1 ) = W1( 1 )
    W2( 2 ) = W1( 2 ) + W1( 3 )
 END IF
CASE DEFAULT ! Element 2 has a trivial transformation - no action required
```

```
   END SELECT
   RETURN
   END SUBROUTINE RANGE
```

This completes the supply of information to LANCELOT_solve. The problem may now be solved using the following program. We choose to solve the model subproblem using a preconditioned conjugate gradient scheme, using diagonal preconditioning, a "box" shaped trust-region, calculating an exact Cauchy point at each iteration but not obtaining an accurate minimizer of the model within the box. Furthermore, we start from the initial estimate $x_1 = 0.0$, $x_2 = 0.0$ and $x_3 = 1.5$. As we have no idea of a good estimate for the Lagrange multiplier associated with the constraint (group 6), we set $y_6 = 0$. We observe that the objective function is bounded below by -2.0 within the feasible box and terminate when the norms of the reduced gradient of the Lagrangian function and constraints are smaller than 10^{-5}.

```
   PROGRAM LANCELOT_example
   USE LANCELOT_double                          ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
   REAL ( KIND = wp ), PARAMETER :: infinity = 10.0_wp ** 20
   TYPE ( LANCELOT_control_type ) :: control
   TYPE ( LANCELOT_inform_type ) :: info
   TYPE ( LANCELOT_data_type ) :: data
   TYPE ( LANCELOT_problem_type ) :: prob
   INTEGER :: i, lfuval
   INTEGER, PARAMETER :: n = 3, ng = 6, nel = 3, nnza = 4, nvrels = 7
   INTEGER, PARAMETER :: ntotel = 3, ngpvlu = 0, nepvlu = 0
   INTEGER :: IVAR( n ), ICALCF( nel ), ICALCG( ng )
   REAL ( KIND = wp ) :: Q( n ), XT( n ), DGRAD( n ), FVALUE( ng )
   REAL ( KIND = wp ) :: GVALUE( ng, 3 )
   REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : ) :: FUVALS
   EXTERNAL RANGE, ELFUN, GROUP
! make space for problem data
   prob%n = n ; prob%ng = ng ; prob%nel = nel
   ALLOCATE( prob%ISTADG( prob%ng + 1 ), prob%ISTGPA( prob%ng + 1 ) )
   ALLOCATE( prob%ISTADA( prob%ng + 1 ), prob%ISTAEV( prob%nel + 1 ) )
   ALLOCATE( prob%ISTEPA( prob%nel + 1 ), prob%ITYPEG( prob%ng ) )
   ALLOCATE( prob%KNDOFG( prob%ng ), prob%ITYPEE( prob%nel ) )
   ALLOCATE( prob%INTVAR( prob%nel + 1 ) )
   ALLOCATE( prob%IELING( ntotel ), prob%IELVAR( nvrels ), prob%ICNA( nnza ) )
   ALLOCATE( prob%ISTADH( prob%nel + 1 ), prob%A( nnza ) )
   ALLOCATE( prob%B( prob%ng ), prob%BL( prob%n ), prob%BU( prob%n ) )
   ALLOCATE( prob%X( prob%n ), prob%Y( prob%ng ), prob%C( prob%ng ) )
   ALLOCATE( prob%GPVALU( ngpvlu ), prob%EPVALU( nepvlu ) )
   ALLOCATE( prob%ESCALE( ntotel ), prob%GSCALE( prob%ng ) )
   ALLOCATE( prob%VSCALE( prob%n ) )
   ALLOCATE( prob%INTREP( prob%nel ), prob%GXEQX( prob%ng ) )
   ALLOCATE( prob%GNAMES( prob%ng ), prob%VNAMES( prob%n ) )
! set problem data
   prob%ISTADG = (/ 1, 1, 2, 3, 3, 4, 4 /)
   prob%IELVAR = (/ 2, 1, 3, 2, 3, 1, 2 /)
   prob%ISTAEV = (/ 1, 4, 6, 8 /) ; prob%INTVAR( : nel ) = (/ 2,  2, 2 /)
   prob%IELING = (/ 1, 2, 3 /) ; prob%ICNA = (/ 1, 2, 1, 2 /)
   prob%ISTADA = (/ 1, 2, 2, 2, 3, 3, 5 /)
   prob%KNDOFG = (/ 1, 1, 1, 1, 1, 2 /)
   prob%ITYPEG = (/ 1, 0, 2, 0, 1, 3 /) ; prob%ITYPEE = (/ 1, 2, 2 /)
   prob%ISTGPA = (/ 0, 0, 0, 0, 0, 0 /) ; prob%ISTEPA = (/ 0, 0, 0 /)
   prob%A = (/ 1.0_wp, 1.0_wp, 1.0_wp, 2.0_wp /)
   prob%B = (/ 0.0_wp, 0.0_wp, 0.0_wp, 0.0_wp, 0.0_wp, 1.0_wp /)
   prob%BL = (/ - infinity, -1.0_wp, 1.0_wp /)
   prob%BU = (/ infinity, 1.0_wp, 2.0_wp /)
   prob%GSCALE = (/ 1.0_wp, 1.0_wp, 3.0_wp, 1.0_wp, 2.0_wp, 1.0_wp /)
   prob%ESCALE = (/ 1.0_wp, 1.0_wp, 1.0_wp /)
   prob%VSCALE = (/ 1.0_wp, 1.0_wp, 1.0_wp /)
```

```
   prob%X = (/ 0.0_wp, 0.0_wp, 1.5_wp /) ;  prob%Y( 6 ) = 0.0_wp
   prob%INTREP = (/ .TRUE., .FALSE., .FALSE. /)
   prob%GXEQX = (/ .FALSE., .TRUE., .FALSE., .TRUE., .FALSE., .FALSE. /)
   prob%GNAMES = (/ 'Obj 1     ', 'Obj 2     ', 'Obj 3     ', 'Obj 4     ', &
                    'Obj 5     ', 'Constraint' /)
   prob%VNAMES = (/ 'x 1', 'x 2', 'x 3' /)
! allocate space for FUVALS
   lfuval = prob%nel + 2 * prob%n
   DO i = 1, prob%nel
     lfuval = lfuval + ( prob%INTVAR( i ) * ( prob%INTVAR( i ) + 3 ) ) / 2
   END DO
   ALLOCATE( FUVALS( lfuval ) )
! problem data complete
   CALL LANCELOT_initialize( data, control )     ! Initialize control parameters
   control%maxit = 100 ; control%out = 6 ! ; control%print_level = 1
   control%stopg = 0.00001_wp ; control%stopc = 0.00001_wp
   control%linear_solver = 1
   control%exact_gcp = .FALSE.
   info%status = 0
! solve the problem
   CALL LANCELOT_solve(                                                  &
       prob, RANGE, GVALUE, FVALUE, XT, FUVALS, lfuval, ICALCF, ICALCG, &
       IVAR, Q, DGRAD, control, info, data, ELFUN  = ELFUN , GROUP = GROUP )
! act on return status
   IF ( info%status == 0 ) THEN                  !  Successful return
     WRITE( 6, "( I6, ' iterations. Optimal objective value =', &
    &       ES12.4, /, ' Optimal solution = ', ( 5ES12.4 ) )" ) &
     info%iter, info%obj, prob%X
   ELSE                                          !  Error returns
     WRITE( 6, "( ' LANCELOT_solve exit status = ', I6 ) " ) info%status
   END IF
   CALL LANCELOT_terminate( data, control, info ) !  delete internal workspace
   DEALLOCATE( prob%GNAMES, prob%VNAMES )         !  delete problem space
   DEALLOCATE( prob%VSCALE, prob%ESCALE, prob%GSCALE, prob%INTREP )
   DEALLOCATE( prob%GPVALU, prob%EPVALU, prob%ITYPEG, prob%GXEQX )
   DEALLOCATE( prob%X, prob%Y, prob%C, prob%A, prob%B, prob%BL, prob%BU )
   DEALLOCATE( prob%ISTADH, prob%IELING, prob%IELVAR, prob%ICNA )
   DEALLOCATE( prob%INTVAR, prob%KNDOFG, prob%ITYPEE, prob%ISTEPA )
   DEALLOCATE( prob%ISTADA, prob%ISTAEV, prob%ISTADG, prob%ISTGPA )
   DEALLOCATE( FUVALS )
   END PROGRAM LANCELOT_example
```

This produces the following output.

```
    14 iterations. Optimal objective value = -6.3129E-01
 Optimal solution =   2.4402E-01 -4.0741E-01  1.0000E+00
```

The same effect may be achieved by returning to the user to evaluate element and group values/derivatives. Simply replace the above call to `LANCELOT_solve` by the following do-loop:

```
   DO
     CALL LANCELOT_solve(                                                &
         prob, RANGE, GVALUE, FVALUE, XT, FUVALS, lfuval, ICALCF, ICALCG, &
         IVAR, Q, DGRAD , control, info, data )
     IF ( info%status >= 0 ) EXIT
     IF ( info%status == - 1 .OR. info%status == - 3 .OR. info%status == - 7 ) &
       THEN
       CALL ELFUN ( FUVALS, XT, prob%EPVALU, info%ncalcf, prob%ITYPEE,         &
           prob%ISTAEV, prob%IELVAR, prob%INTVAR, prob%ISTADH, prob%ISTEPA,    &
           ICALCF, prob%nel, prob%nel + 1, prob%ISTAEV( prob%nel + 1 ) - 1,    &
           prob%nel + 1, prob%nel + 1, prob%nel + 1, prob%nel, lfuval, prob%n, &
```

```
        prob%ISTEPA( prob%nel + 1 ) - 1, 1, i )
      IF ( i /= 0 ) THEN ; info%status = - 11 ; CYCLE ; END IF
    END IF
    IF ( info%status == - 1 .OR. info%status == - 5 .OR. info%status == - 6 ) &
      THEN
      CALL ELFUN ( FUVALS, XT, prob%EPVALU, info%ncalcf, prob%ITYPEE,          &
          prob%ISTAEV, prob%IELVAR, prob%INTVAR, prob%ISTADH, prob%ISTEPA,     &
          ICALCF, prob%nel, prob%nel + 1, prob%ISTAEV( prob%nel + 1 ) - 1,     &
          prob%nel + 1, prob%nel + 1, prob%nel + 1, prob%nel, lfuval, prob%n, &
          prob%ISTEPA( prob%nel + 1 ) - 1, 3, i )
      IF ( i /= 0 ) THEN ; info%status = - 11 ; CYCLE ; END IF
    END IF
    IF ( info%status == - 2 .OR. info%status == - 4 ) THEN
      CALL GROUP ( GVALUE, prob%ng, FVALUE, prob%GPVALU, info%ncalcg,       &
          prob%ITYPEG, prob%ISTGPA, ICALCG, prob%ng, prob%ng + 1, prob%ng, &
          prob%ng, prob%ISTGPA( prob%ng + 1 ) - 1, .FALSE., i )
      IF ( i /= 0 ) THEN ; info%status = - 11 ; CYCLE ; END IF
    END IF
    IF ( info%status == - 2 .OR. info%status == - 5 ) THEN
      CALL GROUP( GVALUE, prob%ng, FVALUE, prob%GPVALU, info%ncalcg,        &
          prob%ITYPEG, prob%ISTGPA, ICALCG, prob%ng, prob%ng + 1, prob%ng, &
          prob%ng, prob%ISTGPA( prob%ng + 1 ) - 1, .TRUE., i )
      IF ( i /= 0 ) THEN ; info%status = - 11 ; CYCLE ; END IF
    END IF
  END DO
```

If the user (for whatever reason) wishes to be more flexible with the derivatives used, the optional argument `ELDERS` must be included and filled appropriately. For example, suppose that we do not want to provide second derivatives for any elements of type 2 (but to use the PSB approximation instead) and no derivatives at all for those of type 1 (but finite-difference gradients and DFP approximate second derivatives). Then (refering back to `ITYPEE`, we shoud set `ELDERS` as

i	1	2	3
ELDERS(1,i)	1	0	0
ELDERS(2,i)	2	3	3

In this case, `ELFUN` should also be replaced by a subroutine `ELFUN_flexible` of the following form:

```
  SUBROUTINE ELFUN_flexible( FUVALS, XVALUE, EPVALU, ncalcf, ITYPEE, ISTAEV, &
                             IELVAR, INTVAR, ISTADH, ISTEPA, ICALCF, ltypee, &
                             lstaev, lelvar, lntvar, lstadh, lstepa, lcalcf, &
                             lfuval, lxvalu, lepvlu, llders, ifflag, ELDERS, &
                             ifstat )
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
  INTEGER, INTENT( IN ) :: ncalcf, ifflag, ltypee, lstaev, lelvar, lntvar
  INTEGER, INTENT( IN ) :: lstadh, lstepa, lcalcf, lfuval, lxvalu, lepvlu
  INTEGER, INTENT( IN ) :: llders
  INTEGER, INTENT( OUT ) :: ifstat
  INTEGER, INTENT( IN ) :: ITYPEE( LTYPEE ), ISTAEV( LSTAEV ), IELVAR( LELVAR )
  INTEGER, INTENT( IN ) :: INTVAR( LNTVAR ), ISTADH( LSTADH ), ISTEPA( LSTEPA )
  INTEGER, INTENT( IN ) :: ICALCF( LCALCF ), ELDERS( 2, llders )
  REAL ( KIND = wp ), INTENT( IN ) :: XVALUE( LXVALU )
  REAL ( KIND = wp ), INTENT( IN ) :: EPVALU( LEPVLU )
  REAL ( KIND = wp ), INTENT( INOUT ) :: FUVALS( LFUVAL )
  INTEGER :: ielemn, ieltyp, ihstrt, ilstrt, igstrt, ipstrt, jcalcf
  REAL ( KIND = wp ) :: v1, v2, u1, u2, u3, cs, sn
  ifstat = 0
  DO jcalcf = 1, ncalcf
    ielemn = ICALCF( jcalcf )
    ilstrt = ISTAEV( ielemn ) - 1
    igstrt = INTVAR( ielemn ) - 1
```

```
     ipstrt = ISTEPA( ielemn ) - 1
     IF ( ELDERS( 2, ielemn ) <= 0 ) ihstrt = ISTADH( ielemn ) - 1
     ieltyp = ITYPEE( ielemn )
     SELECT CASE ( ieltyp )
     CASE ( 1 )
       u1 = XVALUE( IELVAR( ilstrt + 1 ) )
       u2 = XVALUE( IELVAR( ilstrt + 2 ) )
       u3 = XVALUE( IELVAR( ilstrt + 3 ) )
       v1 = u1
       v2 = u2 + u3
       cs = COS( v2 )
       sn = SIN( v2 )
       IF ( ifflag == 1 ) THEN
         FUVALS( ielemn ) = v1 * sn
       ELSE
! no first or second derivatives for element of type 1
       END IF
     CASE ( 2 )
       u1 = XVALUE( IELVAR( ilstrt + 1 ) )
       u2 = XVALUE( IELVAR( ilstrt + 2 ) )
       IF ( ifflag == 1 ) THEN
         FUVALS( ielemn ) = u1 * u2
       ELSE
         IF ( ELDERS( 1, ielemn ) <= 0 ) THEN
           FUVALS( igstrt + 1 ) = u2
           FUVALS( igstrt + 2 ) = u1
! no second derivatives for element of type 2
         END IF
       END IF
     END SELECT
   END DO
   RETURN
   END SUBROUTINE ELFUN_flexible
```