GALAHAD QPC

PACKAGE SPECIFICATION GALAHAD Optimization Library version 2.2

1 SUMMARY

This package uses a crossover method, switching between interior-point and working-set approaches, to solve the **quadratic programming problem**

$$\text{minimize } q(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{g}^T\mathbf{x} + f$$

subject to the general linear constraints

$$c_i^l \le \mathbf{a}_i^T\mathbf{x} \le c_i^u, \quad i = 1,\ldots,m,$$

and the simple bound constraints

$$x_j^l \le x_j \le x_j^u, \quad j = 1,\ldots,n,$$

where the n by n symmetric matrix $\mathbf{H}$, the vectors $\mathbf{g}$, $\mathbf{a}_i$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, $\mathbf{x}^u$ and the scalar f are given. Full advantage is taken of any zero coefficients in the matrix $\mathbf{H}$ or the vectors $\mathbf{a}_i$. Any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite.

If the matrix $\mathbf{H}$ is positive semi-definite, a global solution is found. However, if $\mathbf{H}$ is indefinite, the procedure may find a (weak second-order) critical point that is not the global solution to the given problem.

ATTRIBUTES — Versions: `GALAHAD_QPC_single`, `GALAHAD_QPC_double`. **Uses:** `GALAHAD_CPU_time`, `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_SORT`, `GALAHAD_SILS`, `GALAHAD_QPT`, `GALAHAD_QPP`, `GALAHAD_LSQP`, `GALAHAD_QPA`, `GALAHAD_QPB`, `GALAHAD_SPECFILE`, `GALAHAD_FDC`. **Date:** March 2006. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

Access to the package requires a `USE` statement such as

Single precision version

```
USE GALAHAD_QPC_single
```

Double precision version

```
USE GALAHAD_QPC_double
```

If it is required to use both modules at the same time, the derived types `SMT_type`, `QPT_problem_type`, `QPC_time_type`, `QPC_control_type`, `QPC_inform_type` and `QPC_data_type` (Section 2.2) and the subroutines `QPC_initialize`, `QPC_solve`, `QPC_terminate`, (Section 2.3) and `QPC_read_specfile` (Section 2.5) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

Both the Hessian matrix $\mathbf{H}$ and the constraint Jacobian $\mathbf{A}$, the matrix whose rows are the vectors $\mathbf{a}_i^T$, $i = 1,\ldots,m$, may be stored in a variety of input formats.

**All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.**

2.1.1 Dense storage format

The matrix **A** is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Component $n*(i-1)+j$ of the storage array A%val will hold the value a_{ij} for $i=1,\ldots,m$, $j=1,\ldots,n$. Since **H** is symmetric, only the lower triangular part (that is the part h_{ij} for $1 \leq j \leq i \leq n$) need be held. In this case the lower triangle will be stored by rows, that is component $i*(i-1)/2+j$ of the storage array H%val will hold the value h_{ij} (and, by symmetry, h_{ji}) for $1 \leq j \leq i \leq n$.

2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry of **A**, its row index i, column index j and value a_{ij} are stored in the l-th components of the integer arrays A%row, A%col and real array A%val. The order is unimportant, but the total number of entries A%ne is also required. The same scheme is applicable to **H** (thus requiring integer arrays H%row, H%col, a real array H%val and an integer value H%ne), except that only the entries in the lower triangle need be stored.

2.1.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row $i+1$. For the i-th row of **A**, the i-th component of a integer array A%ptr holds the position of the first entry in this row, while A%ptr $(m+1)$ holds the total number of entries plus one. The column indices j and values a_{ij} of the entries in the i-th row are stored in components $l =$ A%ptr(i), ...,A%ptr $(i+1)-1$ of the integer array A%col, and real array A%val, respectively. The same scheme is applicable to **H** (thus requiring integer arrays H%ptr, H%col, and a real array H%val), except that only the entries in the lower triangle need be stored.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

2.1.4 Diagonal storage format

If **H** is diagonal (i.e., $h_{ij} = 0$ for all $1 \leq i \neq j \leq n$) only the diagonals entries h_{ii}, $1 \leq i \leq n$, need be stored, and the first n components of the array H%val may be used for the purpose. There is no sensible equivalent for the non-square **A**.

2.2 The derived data types

Ten derived data types are accessible from the package.

2.2.1 The derived data type for holding matrices

The derived data type SMT_TYPE is used to hold the matrices **A** and **H**. The components of SMT_TYPE used here are:

m is a scalar component of type default INTEGER, that holds the number of rows in the matrix.

n is a scalar component of type default INTEGER, that holds the number of columns in the matrix.

ne is a scalar variable of type default INTEGER, that holds the number of matrix entries.

type is a rank-one allocatable array of type default CHARACTER, that is used to indicate the matrix storage scheme used. Its precise length and content depends on the type of matrix to be stored (see §2.2.2).

val is a rank-one allocatable array of type default REAL (double precision in GALAHAD_QPC_double) and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $h_{ij} = h_{ji}$ of a *symmetric* matrix **H** is represented as a single entry (see §2.1.1–2.1.3). Any duplicated entries that appear in the sparse co-ordinate or row-wise schemes will be summed.

**All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.**

row is a rank-one allocatable array of type default INTEGER, and dimension at least ne, that may hold the row indices of the entries. (see §2.1.2).

col is a rank-one allocatable array of type default INTEGER, and dimension at least ne, that may the column indices of the entries (see §2.1.2–2.1.3).

ptr is a rank-one allocatable array of type default INTEGER, and dimension at least m + 1, that may holds the pointers to the first entry in each row (see §2.1.3).

2.2.2 The derived data type for holding the problem

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

new_problem_structure is a scalar variable of type default LOGICAL, that is .TRUE. if this is the first (or only) problem in a sequence of problems with identical "structure" to be attempted, and .FALSE. if a previous problem with the same "structure" (but different numerical data) has been solved. Here, the term "structure" refers both to the sparsity patterns of the Jacobian matrices **A** involved (but not their numerical values), to the zero/nonzero/infinity patterns (a bound is either zero, $\pm$ infinity, or a finite but arbitrary nonzero) of each of the constraint bounds, and to the variables and constraints that are fixed (both bounds are the same) or free (the lower and upper bounds are $\pm$ infinity, respectively).

n is a scalar variable of type default INTEGER, that holds the number of optimization variables, n.

m is a scalar variable of type default INTEGER, that holds the number of general linear constraints, m.

H is scalar variable of type SMT_TYPE that holds the Hessian matrix **H**. The following components are used:

H%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1), is used, the first five components of H%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of H%type must contain the string COORDINATE, for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of H%type must contain the string SPARSE_BY_ROWS, and for the diagonal storage scheme (see Section 2.1.4), the first eight components of H%type must contain the string DIAGONAL.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into H%type. For example, if prob is of derived type QPC_problem_type and involves a Hessian we wish to store using the co-ordinate scheme, we may simply

```
CALL SMT_put( prob%H%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

H%ne is a scalar variable of type default INTEGER, that holds the number of entries in the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for any of the other three schemes.

H%val is a rank-one allocatable array of type default REAL (double precision in GALAHAD_QPC_double), that holds the values of the entries of the **lower triangular** part of the Hessian matrix **H** in any of the storage schemes discussed in Section 2.1.

H%row is a rank-one allocatable array of type default INTEGER, that holds the row indices of the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be allocated for any of the other three schemes.

H%col is a rank-one allocatable array variable of type default INTEGER, that holds the column indices of the **lower triangular** part of **H** in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated when the dense or diagonal storage schemes are used.

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

H%ptr is a rank-one allocatable array of dimension n+1 and type default INTEGER, that holds the starting position of each row of the **lower triangular** part of **H**, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3). It need not be allocated when the other schemes are used.

G is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_QPC_double), that holds the gradient $\mathbf{g}$ of the linear term of the quadratic objective function. The j-th component of G, $j = 1,\ldots,n$, contains $\mathbf{g}_j$.

f is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the constant term, f, in the objective function.

A is scalar variable of type SMT_TYPE that holds the Jacobian matrix **A**. The following components are used:

A%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1), is used, the first five components of A%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of A%type must contain the string COORDINATE, while for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of A%type must contain the string SPARSE_BY_ROWS.

Just as for H%type above, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into A%type. Once again, if prob is of derived type QPC_problem_type and involves a Jacobian we wish to store using the sparse row-wise storage scheme, we may simply

```
CALL SMT_put( prob%A%type, 'SPARSE_BY_ROWS' )
```

A%ne is a scalar variable of type default INTEGER, that holds the number of entries in **A** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for either of the other two schemes.

A%val is a rank-one allocatable array of type default REAL (double precision in GALAHAD_QPC_double), that holds the values of the entries of the Jacobian matrix **A** in any of the storage schemes discussed in Section 2.1.

A%row is a rank-one allocatable array of type default INTEGER, that holds the row indices of **A** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be allocated for either of the other two schemes.

A%col is a rank-one allocatable array variable of type default INTEGER, that holds the column indices of **A** in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated when the dense storage scheme is used.

A%ptr is a rank-one allocatable array of dimension m+1 and type default INTEGER, that holds the starting position of each row of **A**, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3). It need not be allocated when the other schemes are used.

C_l is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_QPC_double), that holds the vector of lower bounds $\mathbf{c}^l$ on the general constraints. The i-th component of C_l, $i = 1,\ldots,m$, contains $\mathbf{c}_i^l$. Infinite bounds are allowed by setting the corresponding components of C_l to any value smaller than -infinity, where infinity is a component of the control array control (see Section 2.2.5).

C_u is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_QPC_double), that holds the vector of upper bounds $\mathbf{c}^u$ on the general constraints. The i-th component of C_u, $i = 1,\ldots,m$, contains $\mathbf{c}_i^u$. Infinite bounds are allowed by setting the corresponding components of C_u to any value larger than infinity, where infinity is a component of the control array control (see Section 2.2.5).

X_l is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_QPC_double), that holds the vector of lower bounds $\mathbf{x}^l$ on the the variables. The j-th component of X_l, $j = 1,\ldots,n$, contains $\mathbf{x}_j^l$. Infinite bounds are allowed by setting the corresponding components of X_l to any value smaller than -infinity, where infinity is a component of the control array control (see Section 2.2.5).

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

X_u is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_QPC_double), that holds the vector of upper bounds $\mathbf{x}^u$ on the variables. The j-th component of X_u, $j = 1,\ldots,n$, contains $\mathbf{x}^u_j$. Infinite bounds are allowed by setting the corresponding components of X_u to any value larger than that infinity, where infinity is a component of the control array control (see Section 2.2.5).

X is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_QPC_double), that holds the values $\mathbf{x}$ of the optimization variables. The j-th component of X, $j = 1,\ldots,n$, contains x_j.

Z is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_QPC_double), that holds the values $\mathbf{z}$ of estimates of the dual variables corresponding to the simple bound constraints (see Section 4). The j-th component of Z, $j = 1,\ldots,n$, contains z_j.

C is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_QPC_double), that holds the values $\mathbf{Ax}$ of the constraints. The i-th component of C, $i = 1,\ldots,m$, contains $\mathbf{a}_i^T\mathbf{x} \equiv (\mathbf{Ax})_i$.

Y is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_QPC_double), that holds the values $\mathbf{y}$ of estimates of the Lagrange multipliers corresponding to the general linear constraints (see Section 4). The i-th component of Y, $i = 1,\ldots,m$, contains y_i.

2.2.3 The derived data type for holding control parameters specifically for the interior-point phase

The derived data type QPB_control_type is used to hold controlling data for the interior-point phase of the calculation, performed by the GALAHAD package GALAHAD_QPB. Components may be changed by calling GALAHAD_QPC_read-_spec (see the documentation for GALAHAD_QPB for details). The components of QPC_control_type are:

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

out is a scalar variable of type default INTEGER, that holds the stream number for informational messages. Printing of informational messages in QPB_solve is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type default INTEGER, that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level $= 1$, a single line of output will be produced for each iteration of the process. If print_level ≥ 2, 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 default INTEGER, that holds the maximum number of iterations which will be allowed in QPB_solve. The default is maxit = 1000.

start_print is a scalar variable of type default INTEGER, that specifies the first iteration for which printing will occur in QPB_solve. If start_print is negative, printing will occur from the outset. The default is start_print = -1.

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

factor is a scalar variable of type default INTEGER, that indicates the type of factorization of the preconditioner to be used. Possible values are:

0 the type is chosen automatically on the basis of which option looks likely to be the most efficient.

1 a Schur-complement factorization will be used.

2 an augmented-system factorization will be used.

**All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.**

The default is `factor = 0`.

`max_col` is a scalar variable of type default `INTEGER`, that specifies the maximum number of nonzeros in a column of **A** which is permitted by the Schur-complement factorization. The default is `max_col = 35`.

`indmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of integer workspace required by the factorization package `SILS`. The default is `indmin = 1000`.

`valmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of real workspace required by the factorization package `SILS`. The default is `valmin = 1000`.

`itref_max` is a scalar variable of type default `INTEGER`, that specifies the maximum number of iterative refinements allowed with each application of the preconditioner. The default is `itref_max = 1`.

`infeas_max` is a scalar variable of type default `INTEGER`, that specifies the number of iterations for which the overall infeasibility of the problem is not reduced by at least a factor `reduce_infeas before` the problem is flagged as infeasible (see `reduce_infeas`). The default is `infeas_max = 200`.

`cg_maxit` is a scalar variable of type default `INTEGER`, that holds the maximum number of conjugate-gradient inner iterations that may be performed during the computation of each search direction in `QPB_solve`. If `cg_maxit` is set to a negative number, it will be reset by `QPB_solve` to the dimension of the relevant linear system $+1$. The default is `cg_maxit = 200`.

`precon` is a scalar variable of type default `INTEGER`, that specifies which preconditioner to be used to accelerate the conjugate-gradient inner iteration. Possible values are:

- `0` the type is chosen automatically on the basis of which option looks likely to be the most efficient at any given stage of the solution process. Different preconditioners may be used at different stages.
- `1` the Hessian matrix is replaced by the identity matrix.
- `2` a full factorization using the Hessian, which is equivalent to replacing the conjugate gradient inner iteration by a direct method. The Hessian may be perturbed to ensure that the resultant matrix is a preconditioner.
- `3` the Hessian matrix is replaced by a band of given semi-bandwidth (see `nsemib` below).
- `4` the Hesian matrix is replaced by its barrier terms.

The default is `precon = 0`.

`nsemib` is a scalar variable of type default `INTEGER`, that specifies the semi-bandwidth of the band preconditioner when `precon = 3`, if appropriate. The default is `nsemib = 5`.

`restore_problem` is a scalar variable of type default `INTEGER`, that specifies how much of the input problem is to be retored on output. Possible values are:

- `0` nothing is restored.
- `1` the vector data $\mathbf{g}$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ will be restored to their input values.
- `2` the entire problem, that is the above vector data along with the Hessian matrix **H** and the Jacobian matrix **A**, will be restored.

The default is `restore_problem = 2`.

`infinity` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that is used to specify which constraint bounds are infinite. Any bound larger than `infinity` in modulus will be regarded as infinite. The default is `infinity` $= 10^{19}$.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`. For any commercial application, a separate license must be signed.

stop_p is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the required accuracy for the primal infeasibility (see Section 4). The default is stop_p = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

stop_d is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the required accuracy for the dual infeasibility (see Section 4). The default is stop_d = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

stop_c is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the required accuracy for the violation of complementarity slackness (see Section 4). The default is stop_c = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

prfeas is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that aims to specify the closest that any initial variable may be to infeasibility. Any variable closer to infeasibility than prfeas will be moved to prfeas from the offending bound. However, if a variable is range bounded, and its bounds are closer than prfeas apart, it will be moved to the mid-point of the two bounds. The default is prfeas = 1.0.

dufeas is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that aims to specify the closest that any initial dual variable or Lagrange multiplier may be to infeasibility. Any variable closer to infeasibility than prfeas will be moved to dufeas from the offending bound. However, if a dual variable is range bounded, and its bounds are closer than dufeas apart, it will be moved to the mid-point of the two bounds. The default is dufeas = 1.0.

muzero is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the initial value of the barrier parameter. If muzero is not positive, it will be reset automatically to an appropriate value. The default is muzero = -1.0.

reduce_infeas is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that specifies the least factor by which the overall infeasibility of the problem must be reduced, over infeas_max consecutive iterations, for it not be declared infeasible (see infeas_max). The default is reduce_infeas = 0.99.

obj_unbounded is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that specifies smallest value of the objective function that will be tolerated before the problem is declared to be unbounded from below. The default is potential_unbounded = $-u^{-2}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

pivot_tol is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the threshold pivot tolerence used by the matrix factorization. See the documentation for the package SILS for details. The default is pivot_tol = $u^{3/4}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

pivot_tol_for_dependencies is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the threshold pivot tolerence used by the matrix factorization when attempting to detect linearly dependent constraints. A value larger than pivot_tol is appropriate. See the documentation for the package SILS for details. The default is pivot_tol_for_dependencies = 0.5.

zero_pivot is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double). Any pivots smaller than zero_pivot in absolute value will be regarded to be zero when attempting to detect linearly dependent constraints. The default is zero_pivot = $u^{3/4}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

identical_bounds_tol is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double). Every pair of constraint bounds (c_i^l, c_i^u) or (x_j^l, x_j^u) that is closer than identical_bounds_tol will be reset to the average of their values, $\frac{1}{2}(c_i^l + c_i^u)$ or $\frac{1}{2}(x_j^l + x_j^u)$ respectively. The default is identical_bounds_tol = u, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

`initial_radius` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies the initial trust-region radius. If `initial_radius` is not positive, it will be reset automatically to an appropriate value. The default is `initial_radius = - 1.0`.

`inner_fraction_opt` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies the fraction of the optimal value which is acceptable for the solution of the inner iteration (search direction) problem using the package `GALAHAD_GLTR`, and corresponds to the value `control%fraction_opt` in that package. A negative value is considered to be zero, and a value of larger than one is considered to be one. Reducing `fraction_opt` below one will result in a reduction of the computation performed at the expense of an inferior optimal value. The default is `inner_fraction_opt = 0.1`.

`inner_stop_relative` and `inner_stop_absolute` are scalar variables of type default `REAL` (double precision in `GALAHAD_QPC_double`), that hold the relative and absolute convergence tolerances for the inner iteration (search direction) problem using the package `GALAHAD_GLTR`, and correspond to the values `control%stop_relative` and `control%stop_absolute` in that package. The defaults are `inner_stop_relative = 0.01` and `inner_stop_absolute =` $\sqrt{u}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`cpu_time_limit` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), 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`.

`remove_dependencies` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if the algorithm is to attempt to remove any linearly dependent constraints before solving the problem, and `.FALSE.` otherwise. We recommend removing linearly dependencies. The default is `remove_dependencies = .TRUE.`.

`treat_zero_bounds_as_general` is a scalar variable of type default `LOGICAL`. If it is set to `.FALSE.`, variables which are only bounded on one side, and whose bound is zero, will be recognised as non-negativities/non-positivities rather than simply as lower- or upper-bounded variables. If it is set to `.TRUE.`, any variable bound x_j^l or x_j^u which has the value 0.0 will be treated as if it had a general value. Setting `treat_zero_bounds_as_general` to `.TRUE.` has the advantage that if a sequence of problems are reordered, then bounds which are “accidentally” zero will be considered to have the same structure as those which are nonzero. However, `GALAHAD_QPC` is able to take special advantage of non-negativities/non-positivities, so if a single problem, or if a sequence of problems whose bound structure is known not to change, is/are to be solved, it will pay to set the variable to `.FALSE.`. The default is `treat_zero_bounds_as_general = .FALSE.`.

`center` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if the algorithm will should use the analytic center of the feasible set as its initial feasible point, and `.FALSE.` otherwise. We recommend using the analytic center. The default is `center = .TRUE.`.

`primal` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if a primal-barrier Hessian will be used and `.FALSE.` if the primal-dual Hessian is preferred. We recommend using the primal-dual Hessian. The default is `primal = .FALSE.`.

`feasol` is a scalar variable of type default `LOGICAL`, that should be set `.TRUE.` if the final solution obtained will be perturbed so that variables close to their bounds are moved onto these bounds, and `.FALSE.` otherwise. The default is `feasol = .FALSE.`.

`array_syntax_worse_than_do_loop` is a scalar variable of type default `LOGICAL`, that should be set `.TRUE.` if the compiler is better able to optimize Fortran 77-style do-loops than to exploit Fortran 95-style array syntax when performing vector operations, and `.FALSE.` otherwise. The default is `array_syntax_worse_than_do_loop = .FALSE.`.

`prefix` is a scalar variable of type default `CHARACTER` and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

string `prefix(2:LEN(TRIM(prefix))-1)`, thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default `prefix = ""`.

`LSQP_control` is a scalar variable of type `LSQP_control_type` whose components are used to control the initial feasible point calculation, performed by the package `GALAHAD_LSQP`. See the specification sheet for the package `GALAHAD_LSQP` for details, and appropriate default values (but note that default value for `LSQP_control%feasol` is changed to `.FALSE.`.

2.2.4 The derived data type for holding control parameters specifically for the working-set phase

The derived data type `QPA_control_type` is used to hold controlling data for the working-set phase of the calculation performed by the GALAHAD package `GALAHAD_QPA`. Components may be changed by calling `GALAHAD_QPC_read-_spec` (see the documentation for `GALAHAD_QPA` for details). The components of `QPC_control_type` are:

`error` is a scalar variable of type default `INTEGER`, that holds the stream number for error messages. Printing of error messages in `QPA_solve` and `QPA_terminate` is suppressed if `error` ≤ 0. The default is `error = 6`.

`out` is a scalar variable of type default `INTEGER`, that holds the stream number for informational messages. Printing of informational messages in `QPA_solve` is suppressed if `out` < 0. The default is `out = 6`.

`print_level` is a scalar variable of type default `INTEGER`, that is used to control the amount of informational output which is required. No informational output will occur if `print_level` ≤ 0. If `print_level` $= 1$, a single line of output will be produced for each iteration of the process. If `print_level` ≥ 2, 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 default `INTEGER`, that holds the maximum number of iterations which will be allowed in `QPA_solve`. The default is `maxit = 1000`.

`start_print` is a scalar variable of type default `INTEGER`, that specifies the first iteration for which printing will occur in `QPA_solve`. If `start_print` is negative, printing will occur from the outset. The default is `start_print = -1`.

`stop_print` is a scalar variable of type default `INTEGER`, that specifies the last iteration for which printing will occur in `QPA_solve`. If `stop_print` is negative, printing will occur once it has been started by `start_print`. The default is `stop_print = -1`.

`factor` is a scalar variable of type default `INTEGER`, that indicates the type of factorization of the preconditioner to be used. Possible values are:

`0` the type is chosen automatically on the basis of which option looks likely to be the most efficient.

`1` a Schur-complement factorization will be used.

`2` an augmented-system factorization will be used.

The default is `factor = 0`.

`max_col` is a scalar variable of type default `INTEGER`, that specifies the maximum number of nonzeros in a column of **A** which is permitted by the Schur-complement factorization. The default is `max_col = 35`.

`max_sc` is a scalar variable of type default `INTEGER`, that specifies the maximum number of columns permitted in the Schur complement of the reference matrix (see Section 4) before a refactorization is triggered. The default is `max_sc = 75`.

`indmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of integer workspace required by the factorization package `SILS`. The default is `indmin = 1000`.

**All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.**

`valmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of real workspace required by the factorization package `SILS. The` default is `valmin = 1000`.

`itref_max` is a scalar variable of type default `INTEGER`, that specifies the maximum number of iterative refinements allowed with each application of the preconditioner. The default is `itref_max = 1`.

`cg_maxit` is a scalar variable of type default `INTEGER`, that holds the maximum number of conjugate-gradient inner iterations that may be performed during the computation of each search direction in `QPA_solve`. If `cg_itmax` is set to a negative number, it will be reset by `QPA_solve` to the dimension of the relevant linear system $+1$. The default is `cg_itmax = -1`.

`precon` is a scalar variable of type default `INTEGER`, that specifies which preconditioner to be used to accelerate the conjugate-gradient inner iteration. Possible values are:

- `0` the type is chosen automatically on the basis of which option looks likely to be the most efficient at any given stage of the solution process. Different preconditioners may be used at different stages.
- `1` a full factorization using the Hessian, which is equivalent to replacing the conjugate gradient inner iteration by a direct method. The Hessian may be perturbed to ensure that the resultant matrix is a preconditioner.
- `2` the Hessian matrix is replaced by the identity matrix.
- `3` the Hessian matrix is replaced by a band of given semi-bandwidth (see `nsemib` below).
- `4` the Hessian matrix terms in the current reference matrix (see Section 4) are replaced by the identity matrix.
- `5` the Hessian matrix terms outside a band of given semi-bandwidth in the current reference matrix are replaced by zeros (see `nsemib` below).

The default is `precon = 0`.

`nsemib` is a scalar variable of type default `INTEGER`, that specifies the semi-bandwidth of the band preconditioner when `precon = 3`, if appropriate. The default is `nsemib = 5`.

`full_max_fill` is a scalar variable of type default `INTEGER`. If the ratio of the number of nonzeros in the factors of the reference matrix (see Section 4) to the number of nonzeros in the matrix itself exceeds `full_max_fil`, and the preconditioner is being selected automatically (`precon = 0`), a banded approximation (see `precon = 3`) will be used instead. The default is `full_max_fill = 10`.

`deletion_strategy` is a scalar variable of type default `INTEGER`, that specifies the rules used to determine which constraint to remove from the working set (see Section 4) when necessary to ensure further progress towards the solution. Possible values are:

- `0` the constraint whose Lagrange multiplier most violates its required optimality bound will be removed.
- `1` the most-recently added constraint whose Lagrange multiplier violates its required optimality bound will be removed.
- `k` $>$ `1` among the k most-recently added constraints whose Lagrange multipliers violates their required optimality bounds, the one which most violates its bound will be removed.

The default is `deletion_strategy = 0`.

`restore_problem` is a scalar variable of type default `INTEGER`, that specifies how much of the input problem is to be retored on output. Possible values are:

- `0` nothing is restored.
- `1` the vector data $\mathbf{g}$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ will be restored to their input values.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

2 the entire problem, that is the above vector data along with the Hessian matrix **H** and the Jacobian matrix **A**, will be restored.

The default is `restore_problem = 2`.

`monitor_residuals` is a scalar variable of type default `INTEGER`, that specifies the frequency at which working constraint residuals will be monitored to ensure feasibility. The residuals will be monitored every `monitor_residual` iterations. The default is `monitor_residuals = 1`.

`cold_start` is a scalar variable of type default `INTEGER`, that controls the initial working set (see Section 4)). Possible values are:

0 a "warm start" will be performed. The values set in `C_stat` and `B_stat` indicate which constraints will be included in the initial working set. (see `C_stat` and `B_stat` in Section 2.3.2).

1 the constraints "active" at `p%X` (see Section 2.3.2) will determine the initial working set.

2 the initial working set will be empty.

3 the initial working set will only contain equality constraints.

4 the initial working set will contain as many active constraints as possible, chosen (in order) from equality constraints, simple bounds, and finally general linear constraints.

The default is `cold_start = 3`.

`infeas_check_interval` is a scalar variable of type default `INTEGER`, that gives the number of iterations that are permitted before the infeasibility is checked for improvement. The default is `infeas_check_interval = 100`.

`infinity` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that is used to specify which constraint bounds are infinite. Any bound larger than `infinity` in modulus will be regarded as infinite. The default is `infinity` $= 10^{19}$.

`feas_tol` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies the maximum amount by which a constraint may be violated and yet still be considered to be satisfied. The default is `feas_tol` $= u^{3/4}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`obj_unbounded` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies smallest value of the objective function that will be tolerated before the problem is declared to be unbounded from below. The default is `potential_unbounded =` $-u^{-2}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`increase_rho_g_factor` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that gives the factor by which the current penalty parameter ρ_g for the general constraints may be increased when solving quadratic programs. The default is `increase_rho_g_factor = 2`.

`increase_rho_b_factor` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that gives the factor by which the current penalty parameter ρ_b for the simple bound constraints may be increased when solving quadratic programs. The default is `increase_rho_b_factor = 2`.

`infeas_g_improved_by_factor` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies the relative improvement in the infeasibility that must be achieved when solving quadratic programs if the current value of ρ_g is to be maintained. Specifically if the infeasibility of the general constraints has not fallen by at least a factor `infeas_g_improved_by_factor` during the previous `infeas_check_interval` iterations, the penalty parameter will be increased by a factor `increase_rho_g_factor`. The default is `infeas_improved_g_by_factor = 0.75`.

**All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.**

`infeas_b_improved_by_factor` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that specifies the relative improvement in the infeasibility that must be achieved when solving quadratic programs if the current value of ρ_b is to be maintained. Specifically if the infeasibility of the simple bound constraints has not fallen by at least a factor `infeas_b_improved_by_factor` during the previous `infeas_check_interval` iterations, the penalty parameter will be increased by a factor `increase_rho_b_factor`. The default is `infeas_improved_b_by_factor = 0.75`.

`pivot_tol` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the threshold pivot tolerence used by the matrix factorization. See the documentation for the package `SILS` for details. The default is `pivot_tol` $= 0.1\sqrt{u}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`pivot_tol_for_dependencies` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the threshold pivot tolerence used by the matrix factorization when attempting to detect linearly dependent constraints. A value larger than `pivot_tol` is appropriate. See the documentation for the package `SILS` for details. The default is `pivot_tol_for_dependencies = 0.5`.

`zero_pivot` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`). Any pivots smaller than `zero_pivot` in absolute value will be regarded to be zero when attempting to detect linearly dependent constraints. The default is `zero_pivot` $= u^{3/4}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`multiplier_tol` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`). Any dual variable or Lagrange multiplier which is less than `multiplier_tol` outside its optimal interval will be regarded as being acceptable when checking for optimality. The default is `zero_pivot` $= \sqrt{u}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`inner_stop_relative` and `inner_stop_absolute` are scalar variables of type default `REAL` (double precision in `GALAHAD_QPC_double`), that hold the relative and absolute convergence tolerances for the inner iteration (search direction) problem. and correspond to the values `control%stop_relative` and `control%stop_absolute` in the GALAHAD package `GALAHAD_GLTR`. The defaults are `inner_stop_relative = 0.0` and `inner_stop_absolute` $= \sqrt{u}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`cpu_time_limit` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), 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`.

`treat_zero_bounds_as_general` is a scalar variable of type default `LOGICAL`. If it is set to `.FALSE.`, variables which are only bounded on one side, and whose bound is zero, will be recognised as non-negativities/non-positivities rather than simply as lower- or upper-bounded variables. If it is set to `.TRUE.`, any variable bound x_j^l or x_j^u which has the value 0.0 will be treated as if it had a general value. Setting `treat_zero_bounds_as_general` to `.TRUE.` has the advantage that if a sequence of problems are reordered, then bounds which are "accidentally" zero will be considered to have the same structure as those which are nonzero. However, `GALAHAD_QPC` is able to take special advantage of non-negativities/non-positivities, so if a single problem, or if a sequence of problems whose bound structure is known not to change, is/are to be solved, it will pay to set the variable to `.FALSE.`. The default is `treat_zero_bounds_as_general = .FALSE.`.

`ramdomize` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` if the user wishes to perturb the constraint bounds by small random quantities during the first stage of the solution process, and `.FALSE.` otherwise. Any randomization will ultimately be removed. Randomization helps when solving degenerate problems and is usually to be recommended. The default is `randomize = .TRUE.`.

`array_syntax_worse_than_do_loop` is a scalar variable of type default `LOGICAL`, that should be set `.TRUE.` if the compiler is better able to optimize Fortran 77-style do-loops than to exploit Fortran 95-style array syntax when performing vector operations, and `.FALSE.` otherwise. The default is `array_syntax_worse_than_do_loop = .FALSE.`.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

`prefix` is a scalar variable of type default `CHARACTER` and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string `prefix(2:LEN(TRIM(prefix))-1)`, thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default `prefix = ""`.

2.2.5 The derived data type for holding control parameters

The derived data type `QPC_control_type` is used to hold controlling data. Default values may be obtained by calling `QPC_initialize` (see Section 2.3.1), while components may also be changed by calling `GALAHAD_QPC_read_spec` (see Section 2.5.1). The components of `QPC_control_type` are:

`error` is a scalar variable of type default `INTEGER`, that holds the stream number for error messages. Printing of error messages in `QPC_solve` and `QPC_terminate` is suppressed if `error` ≤ 0. The default is `error = 6`.

`out` is a scalar variable of type default `INTEGER`, that holds the stream number for informational messages. Printing of informational messages in `QPC_solve` is suppressed if `out` < 0. The default is `out = 6`.

`print_level` is a scalar variable of type default `INTEGER`, that is used to control the amount of informational output which is required. No informational output will occur if `print_level` ≤ 0. If `print_level` $= 1$, a single line of output will be produced for each iteration of the process. If `print_level` ≥ 2, this output will be increased to provide significant detail of each iteration. The default is `print_level = 0`.

`indmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of integer workspace required by the factorization package `SILS`. The default is `indmin = 1000`.

`valmin` is a scalar variable of type default `INTEGER`, that specifies an initial estimate as to the amount of real workspace required by the factorization package `SILS`. The default is `valmin = 1000`.

`restore_problem` is a scalar variable of type default `INTEGER`, that specifies how much of the input problem is to be retored on output. Possible values are:

`0` nothing is restored.

`1` the vector data $\mathbf{g}$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ will be restored to their input values.

`2` the entire problem, that is the above vector data along with the Hessian matrix $\mathbf{H}$ and the Jacobian matrix $\mathbf{A}$, will be restored.

The default is `restore_problem = 2`.

`infinity` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that is used to specify which constraint bounds are infinite. Any bound larger than `infinity` in modulus will be regarded as infinite. The default is `infinity` $= 10^{19}$.

`identical_bounds_tol` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`). Every pair of constraint bounds (c_i^l, c_i^u) or (x_j^l, x_j^u) that is closer than `identical_bounds_tol` will be reset to the average of their values, $\frac{1}{2}(c_i^l + c_i^u)$ or $\frac{1}{2}(x_j^l + x_j^u)$ respectively. The default is `identical_bounds_tol` $= u$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_QPC_double`).

`rho_g` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the initial values of the parameter, ρ_g, used to weight the infeasibility term $v_g(\mathbf{x})$ for the working-set phase (see Section 4).

`rho_b` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the initial values of the parameter, ρ_b, used to weight the infeasibility term $v_b(\mathbf{x})$ for the working-set phase (see Section 4).

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

pivot_tol_for_dependencies is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the threshold pivot tolerence used by the matrix factorization when attempting to detect linearly dependent constraints. A value larger than pivot_tol is appropriate. See the documentation for the package SILS for details. The default is pivot_tol_for_dependencies = 0.5.

zero_pivot is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double). Any pivots smaller than zero_pivot in absolute value will be regarded to be zero when attempting to detect linearly dependent constraints. The default is zero_pivot = $u^{3/4}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_QPC_double).

cpu_time_limit is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), 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.

treat_zero_bounds_as_general is a scalar variable of type default LOGICAL. If it is set to .FALSE., variables which are only bounded on one side, and whose bound is zero, will be recognised as non-negativities/non-positivities rather than simply as lower- or upper-bounded variables. If it is set to .TRUE., any variable bound x_j^l or x_j^u which has the value 0.0 will be treated as if it had a general value. Setting treat_zero_bounds_as_general to .TRUE. has the advantage that if a sequence of problems are reordered, then bounds which are "accidentally" zero will be considered to have the same structure as those which are nonzero. However, GALAHAD_QPC is able to take special advantage of non-negativities/non-positivities, so if a single problem, or if a sequence of problems whose bound structure is known not to change, is/are to be solved, it will pay to set the variable to .FALSE.. The default is treat_zero_bounds_as_general = .FALSE..

array_syntax_worse_than_do_loop is a scalar variable of type default LOGICAL, that should be set .TRUE. if the compiler is better able to optimize Fortran 77-style do-loops than to exploit Fortran 95-style array syntax when performing vector operations, and .FALSE. otherwise. The default is array_syntax_worse_than_do_loop = .FALSE..

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..

no_qpa is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to skip the working-set phase of the computation, and .FALSE. if this phase is required. The default is no_qpa = .FALSE..

no_qpb is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to skip the interior-point phase of the computation, and .FALSE. if this phase is required. The default is no_qpa = .FALSE..

qpb_or_qpa is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to use the interior-point phase of the computation, but only to follow this with the working-set phase if the former is unsuccessful, and to .FALSE. if both phases are to be used (subject to the requests made in no_qpa and no_qpb). The default is qpb_or_qpa = .FALSE..

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

QPA_control is a scalar variable of type QPA_control_type whose components are used to control the working-set phase of the calculation, performed by the package GALAHAD_QPA. See the specification sheet for the package GALAHAD_QPA for details, and appropriate default values (but note that default value for QPA_control%feasol is changed to .FALSE..

QPB_control is a scalar variable of type QPB_control_type whose components are used to control the interior-point phase of the calculation, performed by the package GALAHAD_QPB. See the specification sheet for the package GALAHAD_QPB for details, and appropriate default values (but note that default value for QPB_control%feasol is changed to .FALSE..

2.2.6 The derived data type for holding timing information

The derived data type QPC_time_type is used to hold elapsed CPU times for the various parts of the calculation. The components of QPC_time_type are:

total is a scalar variable of type default REAL, that gives the total time spent in the package.

preprocess is a scalar variable of type default REAL, that gives the time spent reordering the problem to standard form prior to solution.

find_dependent is a scalar variable of type default REAL, that gives the time spent detecting and removing linearly-dependent equality constraints

analyse is a scalar variable of type default REAL, that gives the time spent analysing the required matrices prior to factorization.

factorize is a scalar variable of type default REAL, that gives the time spent factorizing the required matrices.

solve is a scalar variable of type default REAL, that gives the time spent computing the search direction.

2.2.7 The derived data type for holding informational parameters specifically from the working-set phase

The derived data type QPA_inform_type is used to hold parameters that give information about the progress of the working-set phase of the algorithm performed by the GALAHAD package GALAHAD_QPA. The components of QPC_inform_type are:

status is a scalar variable of type default INTEGER, that gives the exit status of the algorithm. See Section 2.4 for details.

alloc_status is a scalar variable of type default INTEGER, that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

major_iter is a scalar variable of type default INTEGER, that gives the total number of major iterations required.

iter is a scalar variable of type default INTEGER, that gives the total number of iterations required.

cg_iter is a scalar variable of type default INTEGER, that gives the total number of conjugate-gradient inner iterations required.

factorization_status is a scalar variable of type default INTEGER, that gives the return status from the matrix factorization.

**All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.**

`factorization_integer` is a scalar variable of type default `INTEGER`, that gives the amount of integer storage used for the matrix factorization.

`factorization_real` is a scalar variable of type default `INTEGER`, that gives the amount of real storage used for the matrix factorization.

`nfacts` is a scalar variable of type default `INTEGER`, that gives the total number of factorizations performed.

`nmods` is a scalar variable of type default `INTEGER`, that gives the total number of factorizations which were modified to ensure that the matrix is an appropriate preconditioner.

`num_g_infeas` is a scalar variable of type default `INTEGER`, that gives the total number of general constraints that are violated.

`num_b_infeas` is a scalar variable of type default `INTEGER`, that gives the total number of simple bound constraints that are violated.

`obj` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the value of the objective function at the best estimate of the solution found.

`infeas_g` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the value of the infeasibility $v_g(\mathbf{x})$.

`infeas_b` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the value of the infeasibility $v_b(\mathbf{x})$.

`merit` is a scalar variable of type default `REAL` (double precision in `GALAHAD_QPC_double`), that holds the value of the merit function $q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x})$ at the best estimate of the solution found.

`time` is a scalar variable of type `QPC_time_type` whose components are used to hold elapsed CPU times for the various parts of the calculation (see Section 2.2.6).

2.2.8 The derived data type for holding informational parameters specifically from the working-set phase

The derived data type `QPB_inform_type` is used to hold parameters that give information about the progress of the working-set phase of the algorithm performed by the GALAHAD package `GALAHAD_QPB`. The components of `QPC_inform_type` are:

`status` is a scalar variable of type default `INTEGER`, that gives the exit status of the algorithm. See Section 2.4 for details.

`alloc_status` is a scalar variable of type default `INTEGER`, that gives the status of the last attempted array allocation or deallocation. This will be 0 if `status = 0`.

`bad_alloc` is a scalar variable of type default `CHARACTER` and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if `status = 0`.

`iter` is a scalar variable of type default `INTEGER`, that gives the total number of iterations required.

`factorization_status` is a scalar variable of type default `INTEGER`, that gives the return status from the matrix factorization.

`factorization_integer` is a scalar variable of type default `INTEGER`, that gives the amount of integer storage used for the matrix factorization.

`factorization_real` is a scalar variable of type default `INTEGER`, that gives the amount of real storage used for the matrix factorization.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

nfacts is a scalar variable of type default INTEGER, that gives the total number of factorizations performed.

nbacts is a scalar variable of type default INTEGER, that gives the total number of backtracks performed during the sequence of linesearches.

nmods is a scalar variable of type default INTEGER, that gives the total number of factorizations which were modified to ensure that the matrix is an appropriate preconditioner.

obj is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the value of the objective function at the best estimate of the solution found.

non_negligible_pivot is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the value of the smallest pivot larger than control%zero_pivot when searching for dependent linear constraints. If non_negligible_pivot is close to control%zero_pivot, this may indicate that there are further dependent constraints, and it may be worth increasing control%zero_pivot above non_negligible_pivot and solving again.

feasible is a scalar variable of type default LOGICAL, that has the value .TRUE. if the output value of **x** satisfies the constraints, and the value .FALSE. otherwise.

time is a scalar variable of type QPC_time_type whose components are used to hold elapsed CPU times for the various parts of the calculation (see Section 2.2.6).

2.2.9 The derived data type for holding informational parameters

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

status is a scalar variable of type default INTEGER, that gives the exit status of the algorithm. See Section 2.4 for details.

alloc_status is a scalar variable of type default INTEGER, that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

factorization_status is a scalar variable of type default INTEGER, that gives the return status from the matrix factorization.

factorization_integer is a scalar variable of type default INTEGER, that gives the amount of integer storage used for the matrix factorization.

factorization_real is a scalar variable of type default INTEGER, that gives the amount of real storage used for the matrix factorization.

nfacts is a scalar variable of type default INTEGER, that gives the total number of factorizations performed.

nmods is a scalar variable of type default INTEGER, that gives the total number of factorizations which were modified to ensure that the matrix is an appropriate preconditioner.

obj is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the value of the objective function at the best estimate of the solution found.

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

non_negligible_pivot is a scalar variable of type default REAL (double precision in GALAHAD_QPC_double), that holds the value of the smallest pivot larger than control%zero_pivot when searching for dependent linear constraints. If non_negligible_pivot is close to control%zero_pivot, this may indicate that there are further dependent constraints, and it may be worth increasing control%zero_pivot above non_negligible_pivot and solving again.

p_found is a scalar variable of type default LOGICAL, that has the value .TRUE. if the active-set predictions have been computed, and the value .FALSE. otherwise.

time is a scalar variable of type QPC_time_type whose components are used to hold elapsed CPU times for the various parts of the calculation (see Section 2.2.6).

QPA_inform is a scalar variable of type QPA_inform_type whose components provide information about the progress and needs of the working-set phase of the calculation performed by the package GALAHAD_QPA.

QPB_inform is a scalar variable of type QPB_inform_type whose components provide information about the progress and needs of the interior-point phase of the calculation performed by the package GALAHAD_QPB.

2.2.10 The derived data type for holding problem data

The derived data type QPC_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of QPC procedures. This data should be preserved, untouched, from the initial call to QPC_initialize to the final call to QPC_terminate.

2.3 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.5 for further features):

1. The subroutine QPC_initialize is used to set default values, and initialize private data, before solving one or more problems with the same sparsity and bound structure.

2. The subroutine QPC_solve is called to solve the problem.

3. The subroutine QPC_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by QPC_solve, at the end of the solution process. It is important to do this if the data object is re-used for another problem **with a different structure** since QPC_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We use square brackets [ ] to indicate OPTIONALarguments.

2.3.1 The initialization subroutine

Default values are provided as follows:

```
CALL QPC_initialize( data, control )
```

data is a scalar INTENT(OUT) argument of type QPC_data_type (see Section 2.2.10). It is used to hold data about the problem being solved. QPC_initialize will ensure that all components that are allocatable arrays are disassociated.

control is a scalar INTENT(OUT) argument of type QPC_control_type (see Section 2.2.5). On exit, control contains default values for the components as described in Section 2.2.5. These values should only be changed after calling QPC_initialize.

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

2.3.2 The quadratic programming subroutine

The quadratic programming solution algorithm is called as follows:

```
CALL QPC_solve( p, C_stat, B_stat, data, control, info )
```

p is a scalar INTENT(INOUT) argument of type QPT_problem_type (see Section 2.2.2). It is used to hold data about the problem being solved. For a new problem, the user must allocate all the array components, and set values for all components except p%C. p%new_problem_structure must be set .TRUE., but will have been reset to .FALSE. on exit from QPC_solve. Users are free to choose whichever of the matrix formats described in Section 2.1 is appropriate for **A** and **H** for their application—different formats may be used for the two matrices.

For a problem with the same structure as one that has just been solved, the user may set p%new_problem_structure to .FALSE., so long as QPC_terminate has not been called in the interim. The INTEGER components must be unaltered since the previous call to QPC_solve, but the REAL (double precision in GALAHAD_QPC_double) may be altered to reflect the new problem.

The components p%X, p%Y and p%Z must be set to initial estimates, $\mathbf{x}^0$, of the primal variables, $\mathbf{x}$, Lagrange multipliers for the general constraints, $\mathbf{y}$, and dual variables for the bound constraints, $\mathbf{z}$, respectively. Inappropriate initial values will be altered, so the user should not be overly concerned if suitable values are not apparent, and may be content with merely setting p%X=0.0, p%Y=0.0 and p%Z=0.0. The component p%C need not be set on entry.

On exit, the components p%X, p%Y, p%Z and p%C will contain the best estimates of the primal variables $\mathbf{x}$, Lagrange multipliers for the general constraints $\mathbf{y}$, dual variables for the bound constraints $\mathbf{z}$, and values of the constraints $\mathbf{Ax}$ respectively. What of the remaining problem data has been restored depends upon the input value of the control parameter control%restore_problem. The return format for a restored array component will be the same as its input format. **Restrictions:** p%n > 0, p%m ≥ 0, p%A%ne ≥ -2 and p%H%ne ≥ -2.

C_stat is a rank-one INTENT(INOUT) array argument of dimension p%m and type default INTEGER, that indicates which of the general linear constraints are in the current working set. Possible values for C_stat(i), i$= 1,\ldots,$ p%m, and their meanings are

<0 the i-th general constraint is in the working set, on its lower bound,

>0 the i-th general constraint is in the working set, on its upper bound, and

0 the i-th general constraint is not in the working set.

Suitable values must be supplied if control%qpa_control%cold_start = 0 on entry, but need not be provided for other input values of control%cold_start. Inappropriate values will be ignored. On exit, C_stat will contain values appropriate for the ultimate working set.

B_stat is a rank-one INTENT(INOUT) array argument of dimension p%n and type default INTEGER, that indicates which of the simple bound constraints are in the current working set. Possible values for B_stat(j), j$= 1,\ldots,$ p%n, and their meanings are

<0 the j-th simple bound constraint is in the working set, on its lower bound,

>0 the j-th simple bound constraint is in the working set, on its upper bound, and

0 the j-th simple bound constraint is not in the working set.

Suitable values must be supplied if control%qpa_control%cold_start = 0 on entry, but need not be provided for other input values of control%cold_start. Inappropriate values will be ignored. On exit, B_stat will contain values appropriate for the ultimate working set.

data is a scalar INTENT(INOUT) argument of type QPC_data_type (see Section 2.2.10). 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 QPC_initialize.

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.

`control` is a scalar `INTENT(IN)` argument of type `QPC_control_type` (see Section 2.2.5). Default values may be assigned by calling `QPC_initialize` prior to the first call to `QPC_solve`.

`info` is a scalar `INTENT(OUT)` argument of type `QPC_inform_type` (see Section 2.2.9). A successful call to `QPC_solve` is indicated when the component `status` has the value 0. For other return values of `status`, see Section 2.4.

2.3.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL QPC_terminate( data, control, info )
```

`data` is a scalar `INTENT(INOUT)` argument of type `QPC_data_type` exactly as for `QPC_solve`, which must not have been altered **by the user** since the last call to `QPC_initialize`. On exit, array components will have been deallocated.

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

`info` is a scalar `INTENT(OUT)` argument of type `QPC_inform_type` exactly as for `QPC_solve`. Only the component `status` will be set on exit, and a successful call to `QPC_terminate` is indicated when this component `status` has the value 0. For other return values of `status`, see Section 2.4.

2.4 Warning and error messages

A negative value of `info%status` on exit from `QPC_solve` or `QPC_terminate` indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occured. A message indicating the offending array is written on unit `control%error`, and the returned allocation status and a string containing the name of the offending array are held in `inform%alloc_status` and `inform%bad_alloc` respectively.

-2. A deallocation error occured. A message indicating the offending array is written on unit `control%error` and the returned allocation status and a string containing the name of the offending array are held in `inform%alloc_status` and `inform%bad_alloc` respectively.

-3. One of the restrictions `prob%n` > 0 or `prob%m` ≥ 0 or requirements that `prob%A_type` and `prob%H_type` contain its relevant string `'DENSE'`, `'COORDINATE'`, `'SPARSE_BY_ROWS'` or `'DIAGONAL'` has been violated.

-4. The bound constraints are inconsistent.

-5. The constraints appear to have no feasible point.

-7. The objective function appears to be unbounded from below on the feasible set.

-9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component `inform%factor_status`.

-10. The factorization failed; the return status from the factorization package is given in the component `inform%factor_status`.

-16. The problem is so ill-conditioned that further progress is impossible.

-17. The step is too small to make further impact.

-18. Too many iterations have been performed. This may happen if `control%maxit` is too small, but may also be symptomatic of a badly scaled problem.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

`-19.` The CPU time limit has been reached. This may happen if `control%cpu_time_limit` is too small, but may also be symptomatic of a badly scaled problem.

`-23.` An entry from the strict upper triangle of **H** has been specified.

2.5 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable `control` of type `GALAHAD_QPC_control_type` (see Section 2.2.5), by reading an appropriate data specification file using the subroutine `GALAHAD_QPC_read_specfile`. This facility is useful as it allows a user to change `QPC` control parameters without editing and recompiling programs that call `QPC`.

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 `GALAHAD_QPC_read_specfile` must start with a "`BEGIN GALAHAD_QPC`" command and ends with an "`END`" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by QPC_read_specfile .. )
  BEGIN QPC
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by QPC_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "`BEGIN GALAHAD_QPC`" 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 QPC SPECIFICATION
```

and

```
END QPC SPECIFICATION
```

are acceptable. Furthermore, between the "`BEGIN GALAHAD_QPC`" 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 three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). 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 `GALAHAD_QPC_read_specfile` is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is `REWIND`ed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by `GALAHAD_QPC_read_specfile`.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

2.5.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL QPC_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type GALAHAD_QPC_control_type (see Section 2.2.5). Default values should have already been set, perhaps by calling GALAHAD_QPC_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 2.2.5) of control that each affects are given in Table 2.1.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
initial-integer-workspace	%indmin	integer
initial-real-workspace	%valmin	integer
restore-problem-on-output	%restore_problem	integer
infinity-value	%infinity	real
identical-bounds-tolerance	%identical_bounds_tol	real
initial-rho-g	%rho_g	real
initial-rho-b	%rho_b	real
pivot-tolerance-used-for-dependencies	%pivot_tol_for_dependencies	real
zero-pivot-tolerance	%zero_pivot	real
maximum-cpu-time-limit	%cpu_time_limit	real
treat-zero-bounds-as-general	%treat_zero_bounds_as_general	logical
array-syntax-worse-than-do-loop	%array_syntax_worse_than_do_loop	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
space-critical	%space_critical	logical
no-qpa-phase	%no_qpa	logical
no-qpb-phase	%no_qpb	logical

Table 2.1: Specfile commands and associated components of control.

device is a scalar INTENT(IN) argument of type default INTEGER, 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.

2.6 Information printed

If control%print_level is positive, information about the progress of the algorithm will be printed on unit control-%out. If control%print_level $= 1$, a single line of output will be produced for each iteration of the process. For the initial-feasible-point phase, this will include values of the current primal and dual infeasibility, and violation of complementary slackness, the feasibility-phase objective value, the current steplength, the value of the barrier parameter, the number of backtracks in the linesearch and the elapsed CPU time in seconds. Once a suitable feasible point has been found, the iteration is divided into major iterations, at which the barrier parameter is reduced, and minor iterations, and which the barrier function is approximately minimized for the current value of the barrier parameter. For the major iterations, the value of the barrier parameter, the required values of dual feasibility and violation of complementary slackness, and the current constraint infeasibility are reported. Each minor iteration of the optimality

**All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.**

phase results in a line giving the current dual feasibility and violation of complementary slackness, the objective function value, the ratio of predicted to achieved reduction of the objective function, the trust-region radius, the number of backtracks in the linesearch, the number of conjugate-gradient iterations taken, and the elapsed CPU time in seconds.

If `control%print_level` ≥ 2 this output will be increased to provide significant detail of each iteration. This extra output includes residuals of the linear systems solved, and, for larger values of `control%print_level`, values of the primal and dual variables and Lagrange multipliers.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: `QPC_solve` calls the GALAHAD packages `GALAHAD_CPU_time`, `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_SORT`, `GALAHAD_SILS`, `GALAHAD_QPT`, `GALAHAD_QPP`, `GALAHAD_LSQP`, `GALAHAD_QPA`, `GALAHAD_QPB`, `GALAHAD_SPECFILE` and `GALAHAD_FDC`.

Input/output: Output is under control of the arguments `control%error`, `control%out` and `control%print_level`.

Restrictions: `prob%n` > 0, `prob%m` ≥ 0, `prob%A_type` and `prob%H_type` $\in$ { `'DENSE'`, `'COORDINATE'`, `'SPARSE_BY_ROWS'`, `'DIAGONAL'` }.

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

4 METHOD

The required solution $\mathbf{x}$ necessarily satisfies the primal optimality conditions

$$\mathbf{A}\mathbf{x} = \mathbf{c} \tag{4.1}$$

and

$$\mathbf{c}^l \leq \mathbf{c} \leq \mathbf{c}^u, \;\; \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u, \tag{4.2}$$

the dual optimality conditions

$$\mathbf{H}\mathbf{x} + \mathbf{g} = \mathbf{A}^T\mathbf{y} + \mathbf{z}, \;\; \mathbf{y} = \mathbf{y}^l + \mathbf{y}^u \;\text{ and }\; \mathbf{z} = \mathbf{z}^l + \mathbf{z}^u, \tag{4.3}$$

and

$$\mathbf{y}^l \geq 0, \;\; \mathbf{y}^u \leq 0, \;\; \mathbf{z}^l \geq 0 \;\text{ and }\; \mathbf{z}^u \leq 0, \tag{4.4}$$

and the complementary slackness conditions

$$(\mathbf{A}\mathbf{x} - \mathbf{c}^l)^T\mathbf{y}^l = 0, \;\; (\mathbf{A}\mathbf{x} - \mathbf{c}^u)^T\mathbf{y}^u = 0, \;\; (\mathbf{x} - \mathbf{x}^l)^T\mathbf{z}^l = 0 \;\text{ and }\; (\mathbf{x} - \mathbf{x}^u)^T\mathbf{z}^u = 0, \tag{4.5}$$

where the vectors $\mathbf{y}$ and $\mathbf{z}$ are known as the Lagrange multipliers for the general linear constraints, and the dual variables for the bounds, respectively, and where the vector inequalities hold componentwise. Primal-dual interior point methods iterate towards a point that satisfies these conditions by ultimately aiming to satisfy (4.1), (4.3) and (4.5), while ensuring that (4.2) and (4.4) are satisfied as strict inequalities at each stage. Appropriate norms of the amounts by which (4.1), (4.3) and (4.5) fail to be satisfied are known as the primal and dual infeasibility, and the violation of complementary slackness, respectively. The fact that (4.2) and (4.4) are satisfied as strict inequalities gives such methods their other title, namely interior-point methods.

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

The problem is solved in three phases. The goal of the first "initial feasible point" phase is to find a strictly interior point which is primal feasible, that is that (4.1) is satisfied. The GALAHAD package `GALAHAD_LSQP` is used for this purpose, and offers the options of either accepting the first strictly feasible point found, or preferably of aiming for the so-called "analytic center" of the feasible region. Having found such a suitable initial feasible point, the second "optimality" phase ensures that (4.1) remains satisfied while iterating to satisfy accurately (but not exactly) the dual feasibility (4.3) and complementary slackness (4.5) using the GALAHAD package `GALAHAD_QPB`. This optimality phase proceeds by approximately minimizing a sequence of barrier functions

$$q(\mathbf{x}) - \mu \left[\sum_{i=1}^{m} \log(c_i - c_i^l) + \sum_{i=1}^{m} \log(c_i^u - c_i) + \sum_{j=1}^{n} \log(x_j - x_j^l) + \sum_{j=1}^{n} \log(x_j^u - x_j) \right],$$

for an approriate sequence of positive barrier parameters μ converging to zero while ensuring that (4.1) remain satisfied and that $\mathbf{x}$ and $\mathbf{c}$ are strictly interior points for (4.2). Note that terms in the above sumations corresponding to infinite bounds are ignored, and that equality constraints are treated specially. See the documentation of `GALAHAD_QPB`.for more details.

Since `GALAHAD_QPB` may not give a fully accurate solution, the last phase uses the GALAHAD package `GALAHAD_QPA` to refine this solution. This package uses a active/working-set method to solve the related ℓ_1 quadratic programming problem

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x}),$$

where the he infeasibilities

$$v_g(\mathbf{x}) = \sum_{i=1}^{m} \max(c_i^l - \mathbf{a}_i^T \mathbf{x}, 0) + \sum_{i=1}^{m} \max(\mathbf{a}_i^T \mathbf{x} - c_i^u, 0)$$

and

$$v_b(\mathbf{x}) = \sum_{j=1}^{n} \max(x_j^l - x_j, 0) + \sum_{j=1}^{n} \max(x_j - x_j^u, 0),$$

and ρ_g and ρ_b are appropriately chosen positive weights. See the documentation of `GALAHAD_QPA`.for more details.

In order to make the solution as efficient as possible, the variables and constraints are reordered internally by the GALAHAD package `GALAHAD_QPP` prior to solution. In particular, fixed variables, and free (unbounded on both sides) constraints are temporarily removed.

References:

The interior-point and working-set methods are described in detail in

A. R. Conn, N. I. M. Gould, D. Orban and Ph. L. Toint (1999). A primal-dual trust-region algorithm for minimizing a non-convex function subject to general inequality and linear equality constraints. Mathematical Programming **87** 215-249.

N. I. M. Gould and Ph. L. Toint (2001). "An iterative working-set method for large-scale non-convex quadratic programming". *Applied Numerical Mathematics* **43 (1–2)** (2002) 109–128.

5 EXAMPLE OF USE

Suppose we wish to minimize $\frac{1}{2}x_1^2 + x_2^2 + \frac{3}{2}x_3^2 + 4x_1x_3 + 2x_2 + 1$ subject to the the general linear constraints $1 \leq 2x_1 + x_2 \leq 2$ and $x_2 + x_3 = 2$, and simple bounds $-1 \leq x_1 \leq 1$ and $x_3 \leq 2$. Then, on writing the data for this problem as

$$\mathbf{H} = \begin{pmatrix} 1 & & 4 \\ & 2 & \\ 4 & & 3 \end{pmatrix}, \quad \mathbf{g} = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \quad \mathbf{x}^l = \begin{pmatrix} -1 \\ -\infty \\ -\infty \end{pmatrix} \text{ and } \mathbf{x}^u = \begin{pmatrix} 1 \\ \infty \\ 2 \end{pmatrix},$$

All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.

and

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & \\ & 1 & 1 \end{pmatrix}, \quad \mathbf{c}^l = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \text{and} \quad \mathbf{c}^u = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

in sparse co-ordinate format, we may use the following code:

```
! THIS VERSION: GALAHAD 2.1 - 22/03/2007 AT 09:00 GMT.
   PROGRAM GALAHAD_QPC_EXAMPLE
   USE GALAHAD_QPC_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 ( QPT_problem_type ) :: p
   TYPE ( QPC_data_type ) :: data
   TYPE ( QPC_control_type ) :: control
   TYPE ( QPC_inform_type ) :: info
   INTEGER :: s
   INTEGER, PARAMETER :: n = 3, m = 2, h_ne = 4, a_ne = 4
   INTEGER, ALLOCATABLE, DIMENSION( : ) :: C_stat, B_stat
! start problem data
   ALLOCATE( p%G( n ), p%X_l( n ), p%X_u( n ) )
   ALLOCATE( p%C( m ), p%C_l( m ), p%C_u( m ) )
   ALLOCATE( p%X( n ), p%Y( m ), p%Z( n ) )
   ALLOCATE( B_stat( n ), C_stat( m ) )
   p%new_problem_structure = .TRUE.           ! new structure
   p%n = n ; p%m = m ; p%f = 1.0_wp           ! dimensions & objective constant
   p%G = (/ 0.0_wp, 2.0_wp, 0.0_wp /)         ! objective gradient
   p%C_l = (/ 1.0_wp, 2.0_wp /)               ! constraint lower bound
   p%C_u = (/ 2.0_wp, 2.0_wp /)               ! constraint upper bound
   p%X_l = (/ - 1.0_wp, - infinity, - infinity /) ! variable lower bound
   p%X_u = (/ 1.0_wp, infinity, 2.0_wp /)     ! variable upper bound
   p%X = 0.0_wp ; p%Y = 0.0_wp ; p%Z = 0.0_wp ! start from zero
!  sparse co-ordinate storage format
   CALL SMT_put( p%H%type, 'COORDINATE', s )     ! Specify co-ordinate
   CALL SMT_put( p%A%type, 'COORDINATE', s )     ! storage for H and A
   ALLOCATE( p%H%val( h_ne ), p%H%row( h_ne ), p%H%col( h_ne ) )
   ALLOCATE( p%A%val( a_ne ), p%A%row( a_ne ), p%A%col( a_ne ) )
   p%H%val = (/ 1.0_wp, 2.0_wp, 3.0_wp, 4.0_wp /) ! Hessian H
   p%H%row = (/ 1, 2, 3, 3 /)                     ! NB lower triangle
   p%H%col = (/ 1, 2, 3, 1 /) ; p%H%ne = h_ne
   p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
   p%A%row = (/ 1, 1, 2, 2 /)
   p%A%col = (/ 1, 2, 2, 3 /) ; p%A%ne = a_ne
! problem data complete
   CALL QPC_initialize( data, control )       ! Initialize control parameters
   control%infinity = infinity                ! Set infinity
   CALL QPC_solve( p,  C_stat, B_stat, data, control, info )    ! Solve problem
   IF ( info%status == 0 ) THEN               !  Successful return
     WRITE( 6, "( ' QPC: ', I0, ' QPA and ', I0, ' QPB iterations  ', /,       &
    &     ' Optimal objective value =',                                        &
    &       ES12.4, /, ' Optimal solution = ', ( 5ES12.4 ) )" )                &
     info%QPA_inform%iter, info%QPB_inform%iter, info%obj, p%X
   ELSE                                       !  Error returns
     WRITE( 6, "( ' QPC_solve exit status = ', I6 ) " ) info%status
   END IF
   CALL QPC_terminate( data, control, info )  !  delete internal workspace
```

**All use is subject to licence. See `http://galahad.rl.ac.uk/galahad-www/cou.html`.
For any commercial application, a separate license must be signed.**

```
   END PROGRAM GALAHAD_QPC_EXAMPLE
```

This produces the following output:

```
 QPC: 4 QPA and 7 QPB iterations
 Optimal objective value =  5.4459E+00
 Optimal solution =  -5.4054E-02  1.1081E+00  8.9189E-01
```

The same problem may be solved holding the data in a sparse row-wise storage format by replacing the lines

```
!  sparse co-ordinate storage format
...
! problem data complete
```

by

```
! sparse row-wise storage format
   CALL SMT_put( p%H%type, 'SPARSE_BY_ROWS' )  ! Specify sparse-by-row
   CALL SMT_put( p%A%type, 'SPARSE_BY_ROWS' )  ! storage for H and A
   ALLOCATE( p%H%val( h_ne ), p%H%col( h_ne ), p%H%ptr( n + 1 ) )
   ALLOCATE( p%A%val( a_ne ), p%A%col( a_ne ), p%A%ptr( m + 1 ) )
   p%H%val = (/ 1.0_wp, 2.0_wp, 3.0_wp, 4.0_wp /) ! Hessian H
   p%H%col = (/ 1, 2, 3, 1 /)                     ! NB lower triangular
   p%H%ptr = (/ 1, 2, 3, 5 /)                     ! Set row pointers
   p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
   p%A%col = (/ 1, 2, 2, 3 /)
   p%A%ptr = (/ 1, 3, 5 /)                        ! Set row pointers
! problem data complete
```

or using a dense storage format with the replacement lines

```
! dense storage format
   CALL SMT_put( p%H%type, 'DENSE' )  ! Specify dense
   CALL SMT_put( p%A%type, 'DENSE' )  ! storage for H and A
   ALLOCATE( p%H%val( n * ( n + 1 ) / 2 ) )
   ALLOCATE( p%A%val( n * m ) )
   p%H%val = (/ 1.0_wp, 0.0_wp, 2.0_wp, 4.0_wp, 0.0_wp, 3.0_wp /) ! Hessian
   p%A%val = (/ 2.0_wp, 1.0_wp, 0.0_wp, 0.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian
! problem data complete
```

respectively.

If instead **H** had been the diagonal matrix

$$\mathbf{H} = \begin{pmatrix} 1 & & \\ & 0 & \\ & & 3 \end{pmatrix}$$

but the other data is as before, the diagonal storage scheme might be used for **H**, and in this case we would instead

```
   CALL SMT_put( prob%H%type, 'DIAGONAL' )  ! Specify dense storage for H
   ALLOCATE( p%H%val( n ) )
   p%H%val = (/ 1.0_wp, 0.0_wp, 3.0_wp /) ! Hessian values
```

Notice here that zero diagonal entries are stored.

All use is subject to licence. See http://galahad.rl.ac.uk/galahad-www/cou.html.
For any commercial application, a separate license must be signed.