Module 9.1: nag_qp
Linear and Quadratic Programming

`nag_qp` contains a procedure for solving general linear programming and quadratic programming problems.

Contents

Introduction

This module contains two procedures and a derived type as follows.

- `nag_qp_sol` computes a constrained minimum of a linear or quadratic objective function subject to a set of general linear constraints and/or bounds on the variables. It may also be used to find a feasible point for a set of such constraints (in which case the objective function is omitted). It treats all matrices as dense and hence is not intended for large sparse problems.
- `nag_qp_cntrl_init` assigns default values to all the components of a structure of the derived type `nag_qp_cntrl_`*wp*.
- `nag_qp_cntrl_`*wp* may be used to supply optional parameters to `nag_qp_sol`.

Procedure: nag_qp_sol

1 Description

nag_qp_sol is designed to solve a linear or quadratic programming problem — minimizing a linear or quadratic function subject to constraints on the variables.

The problem is assumed to be stated in the following form:

$$\underset{x \in R^n}{\text{minimize}}\, f(x) \;\text{ subject to }\; l \leq \left\{ \begin{array}{c} x \\ Ax \end{array} \right\} \leq u, \tag{1}$$

where $f(x)$ is the (linear or quadratic) objective function, and the constraints are grouped as follows:

n *simple bounds* on the variables x;

n_{L} *linear constraints*, defined by the n_{L} by n constant matrix A.

You must supply an initial estimate of the solution to (1).

It is possible to specify $f(x)$ in a variety of ways depending upon the particular problem to be solved. The available forms for $f(x)$ are listed below, where the prefixes F, L, S and T stand for 'feasible' (point), 'linear' (programming), 'symmetric' (quadratic programming) and 'trapezoidal' (quadratic programming) respectively, and c is an n-element vector.

Problem type	$f(x)$	Vector c	Matrix H
F	Not applicable	Not applicable	Not applicable
L	$c^T x$	May be zero	Not applicable
S	$c^T x + \frac{1}{2} x^T H x$	May be zero	n by n symmetric
T	$c^T x + \frac{1}{2} x^T H^T H x$	May be zero	m by n upper trapezoidal ($m \leq n$)

There is no restriction on H or $H^T H$ apart from symmetry. If the quadratic function is convex, a global minimum is found; otherwise, a local minimum is found. The default problem type is F, in which the objective function is omitted and this procedure attempts to find a feasible point for the set of constraints. Objective functions are selected by using the optional argument prob_type (see Section 3.2).

The defining feature of a quadratic function $f(x)$ is that the second-derivative matrix $\nabla^2 f(x)$ (the *Hessian matrix*) is constant. For problems of type S, $\nabla^2 f(x) = H$; for problems of type T, $\nabla^2 f(x) = H^T H$; and for problems of type L, $\nabla^2 f(x) = 0$. For problems of type S and T, you may supply H either *explicitly* as a matrix (via the optional argument h; see Section 3.2), or *implicitly* in a procedure (via the optional argument qp_hx; see Section 3.2) that computes the product Hx or $H^T Hx$ (as appropriate) for any given vector x.

The simple bounds on the variables and the linear constraints are distinguished from one another for reasons of computational efficiency (although the simple bounds could have been included in the definition of the linear constraints). There may be no linear constraints, in which case the matrix A is empty ($n_{\mathrm{L}} = 0$).

Upper bounds and/or lower bounds can be specified separately for the variables and constraints. An *equality* constraint can be specified by setting $l_i = u_i$. If certain bounds are not present, the associated elements of l and u can be set to special values that will be treated as $-\infty$ or $+\infty$.

In general, a successful run of this procedure will indicate one of three situations: (i) a minimizer has been found; (ii) the algorithm has terminated at a so-called *dead-point*; or (iii) the problem has no bounded solution. If a minimizer is found, and $\nabla^2 f(x)$ is positive definite or positive semi-definite, this procedure will obtain a global minimizer; otherwise, the solution will be a *local* minimizer (which may or may not be a global minimizer). A dead-point is a point at which the necessary conditions for optimality are satisfied but the sufficient conditions are not. At such a point, a feasible direction of decrease may or may not exist, so the point is not necessarily a local solution of the problem. Verification of optimality in such instances requires further information, and is in general an NP-hard problem (see Pardalos and

Schnitger [7]). Termination at a dead-point can occur only if $\nabla^2 f(x)$ is not positive definite. If $\nabla^2 f(x)$ is positive semi-definite the dead-point will be a *weak minimizer* (i.e., with a unique optimal objective value, but an infinite set of optimal x).

Several options are available for controlling the operation of this procedure, covering facilities such as:

printed output, at the end of each iteration and at the final solution;

algorithmic parameters, such as tolerances and iteration limits.

These options are grouped together in the optional argument `control`, which is a structure of the derived type `nag_qp_cntrl_wp`.

The method used by this procedure is described in detail in the Mathematical Background section of this module document. It is most efficient when many bounds or linear constraints are active at the solution.

2 Usage

```
USE nag_qp
CALL nag_qp_sol(x, obj_f  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n \geq 1$ — the number of variables

$n_{\mathrm{L}} \geq 0$ — the number of linear constraints

3.1 Mandatory Arguments

x(n) — real(kind=*wp*), intent(inout)

Input: an initial estimate of the solution.

Output: the point at which this procedure terminated. If `error%code` = 0 or 101, `x` contains an estimate of the solution.

obj_f — real(kind=*wp*), intent(out)

Output: the value of the objective function at x if x is feasible, or the sum of infeasibilities at x otherwise. If `prob_type` = `'F'` (the default; see Section 3.2) and x is feasible, `obj_f` is set to zero.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

prob_type — character(len=1), intent(in), optional

Input: specifies the type of objective function, $f(x)$, to be minimized as follows.

prob_type	$f(x)$	Vector c	Matrix H
'F' or 'f'	Not applicable	Not applicable	Not applicable
'L' or 'l'	$c^T x$	Must be supplied in c	Not applicable
'S' or 's'	$c^T x + \frac{1}{2} x^T H x$	May be zero; used if supplied in c	n by n symmetric; may be supplied either explicitly (in h) or implicitly (via qp_hx)
'T' or 't'	$c^T x + \frac{1}{2} x^T H^T H x$	May be zero; used if supplied in c	m by n upper trapezoidal (where $m \leq n$); may be supplied either explicitly (in h) or implicitly (via qp_hx)

Constraints: prob_type = 'F', 'f', 'L', 'l', 'S', 's', 'T' or 't'.

Default: prob_type = 'F'.

x_lower(n) — real(kind=*wp*), intent(in), optional

x_upper(n) — real(kind=*wp*), intent(in), optional

Input: the lower and upper bounds on all the variables. To specify a non-existent lower bound (i.e., $l_j = -\infty$), set x_lower(j) $\leq -$control%inf_bound; to specify a non-existent upper bound (i.e., $u_j = +\infty$), set x_upper(j) $\geq +$control%inf_bound (see the type definition for nag_qp_cntrl_*wp*).

Constraints:

x_lower(j) $\leq$ x_upper(j) for $j = 1, 2, \ldots, n$;

$|\beta| <$ control%inf_bound when x_lower(j) = x_upper(j) = β.

Default: x_lower = $-$control%inf_bound; x_upper = $+$control%inf_bound.

a(n_L, n) — real(kind=*wp*), intent(in), optional

Input: the ith row of a must contain the coefficients of the ith linear constraint, for $i = 1, 2, \ldots, n_\mathrm{L}$.

Default: the problem contains no linear constraints.

ax_lower(n_L) — real(kind=*wp*), intent(in), optional

ax_upper(n_L) — real(kind=*wp*), intent(in), optional

Input: the lower and upper bounds on all the linear constraints. To specify a non-existent lower bound (i.e., $l_j = -\infty$), set ax_lower(j) $\leq -$control%inf_bound; to specify a non-existent upper bound (i.e., $u_j = +\infty$), set ax_upper(j) $\geq +$control%inf_bound (see the type definition for nag_qp_cntrl_*wp*).

Constraints:

ax_lower and ax_upper must not be present unless a is present;

ax_lower(j) $\leq$ ax_upper(j) for $j = 1, 2, \ldots, n_\mathrm{L}$;

$|\beta| <$ control%inf_bound when ax_lower(j) = ax_upper(j) = β.

Default: ax_lower = $-$control%inf_bound; ax_upper = $+$control%inf_bound.

c(n) — real(kind=*wp*), intent(in), optional

Input: the coefficients of the explicit linear term, c, of the objective function.

Constraints: if prob_type = 'L', c must be present. If prob_type = 'F' (the default), c must not be present.

Default: the objective function does not contain an explicit linear term.

h(n_H, n) — real(kind=*wp*), intent(in), optional

Input: h may be used to store the quadratic term H of the objective function if desired. In some cases, you need not use h to store H explicitly (see the description of the optional procedure qp_hx).

If prob_type = 'S', h must contain the upper triangle of the n_H $(= n)$ by n symmetric Hessian matrix. Elements of the array below the diagonal are not referenced.

If prob_type = 'T', h must contain an n_H $(= m)$ by n upper trapezoidal factor of the Hessian matrix. The factor need not be of full rank, i.e., some of the diagonal elements may be zero. However, as a general rule, the larger the dimension of the leading non-singular sub-matrix of H, the fewer iterations will be required. Elements of the array outside the upper trapezoidal part are not referenced.

Constraints: if prob_type = 'S' or 'T', either h or qp_hx must be present. If prob_type = 'F' (the default) or 'L', h must not be present.

Default: the objective function does not contain an explicit quadratic term unless qp_hx is present.

qp_hx — subroutine, optional

The procedure qp_hx may be used to calculate either the product of H and a vector x if prob_type = 'S', or the product of $H^T H$ and a vector x if prob_type = 'T'.

Its specification is:

```
subroutine qp_hx(col, x, hx)

integer, intent(in) ::  col
```

Input: specifies whether or not the vector x is a column of the identity matrix.

If col $= j > 0$, then the vector x is the jth column of the identity matrix, and hence Hx or $H^T Hx$ is the jth column of H or $H^T H$ respectively, which may in some cases require very little computation and qp_hx may be coded to take advantage of this. However, special code is not necessary because x is always stored explicitly in the array x.

If col $= 0$, x has no special form.

```
real(kind=wp), intent(in) ::  x(:)
```

Shape: x has shape (n).

Input: the vector x.

```
real(kind=wp), intent(inout) ::  hx(:)
```

Shape: hx has shape (n).

Input: the zero vector.

Output: the product Hx if prob_type = 'S' or $H^T Hx$ if prob_type = 'T'.

Constraints: if prob_type = 'S' or 'T', either qp_hx or h must be present. If prob_type = 'F' (the default) or 'L', qp_hx must not be present.

Default: the objective function does not contain an explicit quadratic term unless h is present.

cold_start — logical, intent(in), optional

Input: specifies how the initial working set is chosen.

With a *cold start* (i.e., cold_start = .true.), this procedure chooses the initial working set based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or 'nearly' satisfy their bounds (to within control%crash_tol; see the type definition for nag_qp_cntrl_*wp*).

With a *warm start* (i.e., cold_start = .false.), one or both of the arrays x_state and ax_state must be supplied and initialized. This procedure will override the contents of these arrays if necessary, so that a poor choice of the working set will not cause a fatal error. For instance, any elements of x_state or ax_state which are set to -2, -1 or 4 will be reset to zero. A warm start will be advantageous if a good estimate of the initial working set is available, for example when this procedure is called repeatedly to solve related problems.

Default: cold_start = .true..

x_state(n) — integer, intent(inout), optional

Input: if cold_start = .true. (the default), x_state need not be initialized.

If cold_start = .false., x_state specifies the status of the upper and lower bounds on the variables at the start of the feasibility phase. Possible values for x_state(j) (also used by ax_state) are as follows:

x_state(j)	Meaning
0	The corresponding constraint should *not* be in the initial working set.
1	The constraint should be in the initial working set at its lower bound.
2	The constraint should be in the initial working set at its upper bound.
3	The constraint should be in the initial working set as an equality. This value must not be specified unless the corresponding lower and upper bounds are equal.

The values -2, -1 and 4 are also acceptable but will be reset to zero by this procedure, which also adjusts (if necessary) the values supplied in x to be consistent with x_state. Note that x_state already contains satisfactory information if it was present in a previous call to this procedure with the same value of n. (See also the description of cold_start.)

Constraints: $-2 \leq$ x_state(j) ≤ 4, for $j = 1, 2, \ldots, n$. If cold_start = .false., at least one of x_state and ax_state must be present.

Output: the status of the bound constraints in the working set at the point returned in x. The significance of each possible value of x_state(j) (also used by ax_state) is as follows:

x_state(j)	Meaning
-2	This constraint violates its lower bound by more than the feasibility tolerance.
-1	This constraint violates its upper bound by more than the feasibility tolerance.
0	This constraint is satisfied to within the feasibility tolerance, but is not in the working set.
1	This constraint is included in the working set at its lower bound.
2	This constraint is included in the working set at its upper bound.
3	This constraint is included in the working set as an equality. This can only occur when the corresponding upper and lower bounds are equal.
4	This corresponds to optimality being declared with x(j) being temporarily fixed at its current value. This can only occur when error%code = 101 on exit.

Default: if cold_start = .false., x_state = 0.

ax_state(n_L) — integer, intent(inout), optional

Input: if cold_start = .true. (the default), ax_state need not be initialized.

If cold_start = .false., ax_state specifies the status of the upper and lower bounds on the linear constraints at the start of the feasibility phase. Possible values for ax_state(j) (also used by x_state) are as follows:

ax_state(j)	Meaning
0	The corresponding constraint should *not* be in the initial working set.
1	The constraint should be in the initial working set at its lower bound.
2	The constraint should be in the initial working set at its upper bound.
3	The constraint should be in the initial working set as an equality. This value must not be specified unless the corresponding lower and upper bounds are equal.

The values $-2, -1$ and 4 are also acceptable but will be reset to zero by this procedure, which also adjusts (if necessary) the values supplied in `x` to be consistent with `ax_state`. Note that `ax_state` already contains satisfactory information if it was present in a previous call to this procedure with the same value of n_{L}. (See also the description of `cold_start`.)

Constraints: `ax_state` must not be present unless `a` is present and $-2 \leq$ `ax_state`$(j) \leq 4$, for $j = 1, 2, \ldots, n_{\mathrm{L}}$. If `cold_start` = `.false.`, at least one of `ax_state` and `x_state` must be present.

Output: the status of the linear constraints in the working set at the point returned in `x`. The significance of each possible value of `ax_state`(j) (also used by `x_state`) is as follows:

`ax_state`(j)	Meaning
-2	This constraint violates its lower bound by more than the feasibility tolerance.
-1	This constraint violates its upper bound by more than the feasibility tolerance.
0	This constraint is satisfied to within the feasibility tolerance, but is not in the working set.
1	This constraint is included in the working set at its lower bound.
2	This constraint is included in the working set at its upper bound.
3	This constraint is included in the working set as an equality. This can only occur when the corresponding upper and lower bounds are equal.

Default: if `cold_start` = `.false.`, `ax_state` = 0.

x_lambda(n) — real(kind=*wp*), intent(out), optional

Output: the values of the Lagrange multipliers for the bound constraints on the variables with respect to the current working set. If `x_state`$(j) = 0$, (i.e., constraint j is not in the working set), `x_lambda`(j) is zero. If x is optimal, `x_lambda`(j) should be non-negative if `x_state`$(j) = 1$, non-positive if `x_state`$(j) = 2$ and zero if `x_state`$(j) = 4$.

ax_lambda(n_{L}) — real(kind=*wp*), intent(out), optional

Output: the values of the Lagrange multipliers for the linear constraints with respect to the current working set. If `ax_state`$(j) = 0$, (i.e., constraint j is not in the working set), `ax_lambda`(j) is zero. If x is optimal, `ax_lambda`(j) should be non-negative if `ax_state`$(j) = 1$, non-positive if `ax_state`$(j) = 2$.

Constraints: `ax_lambda` must not be present unless `a` is present.

iter — integer, intent(out), optional

Output: the total number of iterations performed.

ax(n_{L}) — real(kind=*wp*), intent(out), optional

Output: the final values of the linear constraints Ax.

Constraints: `ax` must not be present unless `a` is present.

control — type(nag_qp_cntrl_*wp*), intent(in), optional

Input: a structure containing scalar components; these are used to alter the default values of those parameters which control the behaviour of the algorithm and level of printed output. The initialization of this structure and its use is described in the procedure document for `nag_qp_cntrl_init`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The solution appears to be unbounded, i.e., the objective function is not bounded below in the feasible region. This occurs if a step larger than `control%inf_bound` (default value = 10^{20}; see the type definition for `nag_qp_cntrl_`*wp*) would have to be taken in order to continue the algorithm, or the next step would result in an element of x having magnitude larger than `control%inf_step` (default value = max(`control%inf_bound`, 10^{20})).
202	No feasible point was found, i.e., it was not possible to satisfy all the constraints to within the feasibility tolerance. In this case, the constraint violations at the final x will reveal a value of the tolerance for which a feasible point will exist, for example when the feasibility tolerance for each violated constraint exceeds its `Slack` (see Section 7) at the final point. The modified problem (with an altered feasibility tolerance) may then be solved using `cold_start = .false.` (see Section 3.2). You should also check that there are no constraint redundancies. If the data for the constraints are accurate only to the absolute precision σ, you should ensure that the value of `control%feas_tol` (default value = `SQRT(EPSILON(1.0_`*wp*`))`; see the type definition for `nag_qp_cntrl_`*wp*) is *greater* than σ. For example, if all the elements of A are of order unity and are accurate only to three decimal places, then `control%feas_tol` should be at least 10^{-3}.
203	The reduced Hessian exceeds its assigned dimension. The algorithm needed to expand the reduced Hessian when it was already at its maximum dimension, as specified by `control%max_deg_free` (default value = n or `SIZE(h,1)`; see the type definition for `nag_qp_cntrl_`*wp*). The value of `control%max_deg_free` is too small. Rerun this procedure with a larger value (possibly using `cold_start = .false.` (see Section 3.2) to specify the initial working set).

Warnings (error%level = 1):

error%code	Description
101	The iterations were terminated at a dead-point. The necessary conditions for optimality are satisfied but the sufficient conditions are not. (The reduced gradient is negligible, the Lagrange multipliers are optimal, but H_R is singular or there are some very small multipliers.) If $\nabla^2 f(x)$ is not positive definite, x is not necessarily a local solution of the problem and verification of optimality

requires further information. If $\nabla^2 f(x)$ is positive semi-definite or `prob_type` = `'L'` (see Section 3.2), x gives the global minimum value of the objective function, but the final x is not unique.

102 The limiting number of iterations was reached before normal termination occurred.

The values of `control%feas_phase_iter_lim` (default value = max(50, 5 × $(n + n_L)$); see the type definition for `nag_qp_cntrl_wp`) and (if appropriate) `control%opt_phase_iter_lim` (default value = max(50, 5 × $(n + n_L)$) may be too small. If the method appears to be making progress (e.g., the objective function is being satisfactorily reduced), either increase the iterations limit and rerun this procedure or, alternatively, rerun this procedure using `cold_start` = `.false.` (see Section 3.2) to specify the initial working set.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 to 4 of this module document.

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. The following example illustrates the use of the optional arguments `prob_type`, `c`, `x_lower`, `a`, `ax_upper` and `control` in order to solve the linear programming (LP) problem

$$\text{minimize } c^T x \text{ subject to } x \geq l \text{ and } Ax \leq u.$$

The value of `prob_type` specifies that an LP problem is being solved and `control%print_level` = 1 suppresses all intermediate output. The remaining optional arguments are used to store the problem data as follows. The matrix A is stored in the array `a` and the vectors c, l and u are stored in the arrays `c`, `x_lower` and `ax_upper` respectively.

```
   ...
! Initialize control structure
CALL nag_qp_cntrl_init(control)

! Set required value
control%print_level = 1

CALL nag_qp(x, obj_f, prob_type='L', x_lower=x_lower, &
            c=c, a=a, ax_upper=ax_upper, control=control)
   ...
```

6 Further Comments

6.1 Scaling

Sensible scaling of the problem is likely to reduce the number of iterations required and make the problem less sensitive to perturbations in the data, thus improving the condition of the problem. In the absence of better information it usually sensible to make the Euclidean lengths of each constraint of comparable magnitude. See the Chapter Introduction and Gill *et al.* [6] for further information and advice.

6.2 Accuracy

This procedure implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

6.3 Overflow

If the printed output before the overflow error contains a warning about serious ill conditioning in the working set when adding the jth constraint, it may be possible to avoid the difficulty by increasing the magnitude of `control%feas_tol` (default value = `SQRT(EPSILON(1.0_wp))`; see the type definition for `nag_qp_cntrl_wp`) and rerunning the program. If the message recurs even after this change, the offending linearly dependent constraint (with index 'j') *must* be removed from the problem.

7 Description of Printed Output

This section describes the intermediate and final printout produced by this procedure. The level of printed output can be controlled via the components `list` and `print_level` of the optional argument `control`. For example, a listing of the parameter settings to be used by this procedure is output unless `control%list` is set to `.false.`. Note also that the intermediate printout and the final printout are produced only if `control%print_level` ≥ 10 (the default).

When `control%print_level` ≥ 5 and `control%lt80_char` = `.true.` (the default), the following line of output (< 80 characters) is produced at every iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn` is the iteration count.

`Step` is the step taken along the computed search direction. If a constraint is added during the current iteration, `Step` will be the step to the nearest constraint. When `prob_type` = `'L'` (see Section 3.2), the step can be greater than one during the optimality phase.

`Ninf` is the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of the constraint violations. If x is feasible, `Objective` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase, the value of the objective function will be non-increasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Norm Gz` is $||Z_R^T g_{\mathrm{FR}}||$, the Euclidean norm of the reduced gradient with respect to Z_R (see Section 2 and Section 4 of the Mathematical Background section of this module document). During the optimality phase, this norm will be approximately zero after a unit step.

When `control%print_level` ≥ 5 and `control%lt80_char` = `.false.`, the following line of output (up to 120 characters) is produced at every iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration. The following convention is used for numbering the constraints: indices 1 through n refer to the bounds on the variables, and indices $n+1$ through $n+n_{\mathrm{L}}$ refer to the linear constraints (if any). When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

`Itn` (as above)

`Jdel` is the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` is the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` (as above)

`Ninf` (as above)

`Sinf/Objective` (as above)

`Bnd` is the number of simple bound constraints in the current working set.

`Lin` is the number of linear constraints in the current working set.

`Art` is the number of artificial constraints in the working set, i.e., the number of columns of Z_A (see Section 4 of the Mathematical Background section of this module document).

`Zr` is the number of columns of Z_R (see Section 2 of the Mathematical Background section of this module document). `Zr` is the dimension of the sub-space in which the objective function is currently being minimized. The value of `Zr` is the number of variables minus the number of constraints in the working set; i.e., `Zr` $= n$ $-$(`Bnd`+`Lin`+`Art`).
The value of n_Z, the number of columns of Z (see Section 2 of the Mathematical Background section of this module document) can be calculated as $n_Z = n$ $-$(`Bnd`+`Lin`). A zero value of n_Z implies that x lies at a vertex of the feasible region.

`Norm Gz` (as above)

`NOpt` is the number of non-optimal Lagrange multipliers at the current point. `NOpt` is not printed if the current x is infeasible or no multipliers have been calculated. At a minimizer, `NOpt` will be zero.

`Min Lm` is the value of the Lagrange multiplier associated with the deleted constraint. If `Min Lm` is negative, a lower bound constraint has been deleted. If `Min Lm` is positive, an upper bound constraint has been deleted. If no multipliers are calculated during a given iteration, `Min Lm` will be zero.

`Cond T` is a lower bound on the condition number of the working set.

`Cond Rz` is a lower bound on the condition number of the triangular factor R (the Cholesky factor of the current reduced Hessian; see Section 2 of the Mathematical Background section of this module document). If `prob_type` = `'L'` (see Section 3.2), `Cond Rz` is not printed.

`Rzz` is the last diagonal element μ of the matrix D associated with the R^TDR factorization of the reduced Hessian H_R (see Section 3 of the Mathematical Background section of this module document). `Rzz` is only printed if H_R is not positive definite (in which case $\mu \neq 1$). If the printed value of `Rzz` is small in absolute value, then H_R is approximately singular. A negative value of `Rzz` implies that the objective function has negative curvature on the current working set.

The final printout includes a listing of the status of every variable and constraint.

The following describes the printout for each variable. A full stop (.) is printed for any numerical value that is zero.

`Varbl` gives the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State` gives the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than `control%feas_tol` (default value = `SQRT(EPSILON(1.0_`*wp*`))`; see the type definition for `nag_qp_cntrl_`*wp*), `State` will be `++` or `--` respectively.
A key is sometimes printed before `State` to give additional information about the state of a variable.

- `A` *Alternative optimum possible.* The variable is active at one of its bounds, but its Lagrange multiplier is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change to the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them would encounter a bound immediately. In either case the values of the Lagrange multipliers might also change.
- `D` *Degenerate.* The variable is free, but it is equal to (or very close to) one of its bounds.
- `I` *Infeasible.* The variable is currently violating one of its bounds by more than `control%feas_tol`.

`Value`	is the value of the variable at the final iterate.
`Lower Bound`	is the lower bound specified for the variable. `None` indicates that `x_lower`(j) $\leq -$`control%inf_bound` (default value $= 10^{20}$; see the type definition for `nag_qp_cntrl_`*wp*).
`Upper Bound`	is the upper bound specified for the variable. `None` indicates that `x_upper`(j) $\geq$ `control%inf_bound`.
`Lagr Mult`	is the Lagrange multiplier for the associated bound. This will be zero if `State` is `FR` unless `x_lower`(j) $\leq -$`control%inf_bound` and `x_upper`(j) $\geq$ `control%inf_bound`, in which case the entry will be blank. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.
`Slack`	is the difference between the variable `Value` and the nearer of its (finite) bounds `x_lower`(j) and `x_upper`(j). A blank entry indicates that the associated variable is not bounded (i.e., `x_lower`(j) $\leq -$`control%inf_bound` and `x_upper`(j) $\geq$ `control%inf_bound`).

The meaning of the printout for linear constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', `x_lower` and `x_upper` are replaced by `ax_lower` and `ax_upper` respectively, and with the following change in the heading:

`L Con`	gives the name (`L`) and index j, for $j = 1, 2, \ldots, n_{\mathrm{L}}$ of the linear constraint.

Note that movement off a constraint (as opposed to a variable moving away from its bound) can be interpreted as allowing the entry in the `Slack` column to become positive.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

Procedure: nag_qp_cntrl_init

1 Description

nag_qp_cntrl_init assigns default values to the components of a structure of the derived type nag_qp_cntrl_*wp*.

2 Usage

```
USE nag_qp
CALL nag_qp_cntrl_init(control)
```

3 Arguments

3.1 Mandatory Argument

control — type(nag_qp_cntrl_*wp*), intent(out)

Output: a structure containing the default values of those parameters which control the behaviour of the algorithm and level of printed output. A description of its components is given in the document for the derived type nag_qp_cntrl_*wp*.

4 Error Codes

None.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

Derived Type: nag_qp_cntrl_*wp*

Note. The names of derived types containing real/complex components are precision dependent. For double precision the name of this type is nag_qp_cntrl_dp. For single precision the name is nag_qp_cntrl_sp. Please read the Users' Note for your implementation to check which precisions are available.

1 Description

A structure of type nag_qp_cntrl_*wp* is used to supply a number of optional parameters: these govern the level of printed output and a number of tolerances and limits, which allow you to influence the behaviour of the algorithm. If this structure is supplied then it *must* be initialized prior to use by calling the procedure nag_qp_cntrl_init, which assigns default values to all the structure components. You may then assign required values to selected components of the structure (as appropriate).

2 Type Definition

The public components are listed below; components are grouped according to their function. A full description of the purpose of each component is given in Section 3.

```
type nag_qp_cntrl_wp
  !  Printing parameters
  logical ::  list
  integer ::  unit
  logical ::  lt80_char
  integer ::  print_level
  !
  !  Algorithm choice and tolerances
  integer ::  check_freq
  real(kind=wp) ::  crash_tol
  integer ::  expand_freq
  integer ::  feas_phase_iter_lim
  real(kind=wp) ::  feas_tol
  real(kind=wp) ::  inf_bound
  real(kind=wp) ::  inf_step
  integer ::  max_deg_free
  integer ::  opt_phase_iter_lim
  real(kind=wp) ::  rank_tol
  logical ::  min_sum_of_infeas
end type nag_qp_cntrl_wp
```

3 Components

3.1 Printing Parameters

list — logical

Controls the printing of the parameter settings in the call to nag_qp_sol.

If list = .true., then the parameter settings are printed;

if list = .false., then the parameter settings are not printed.

Default: list = .true..

unit — integer

Specifies the Fortran unit number to which all output produced by nag_qp_sol is sent.

Default: unit = the default Fortran output unit number for your implementation.

Constraints: a valid output unit.

lt80_char — logical

Controls the maximum length of each line of output produced by `nag_qp_sol`.

If `lt80_char` = `.true.`, then the output will not exceed 80 characters per line;

if `lt80_char` = `.false.`, then the output will not exceed 120 characters per line whenever `print_level` ≥ 5 (the default).

Default: `lt80_char` = `.true.`.

print_level — integer

Controls the amount of output produced by `nag_qp_sol`, as indicated below. A detailed description of the printed output is given in Section 7 of the procedure document for `nag_qp_sol`.

If `lt80_char` = `.true.` (the default), the following output is sent to the Fortran unit number defined by `unit`:

	0	No output.
	1	The final solution only.
$\geq$	5	One line of summary output (< 80 characters) for each iteration (no printout of the final solution).
$\geq$	10	The final solution and one line of summary output for each iteration.

If `lt80_char` = `.false.`, the following output is sent to the Fortran unit number defined by `unit`:

	0	No output.
	1	The final solution only.
$\geq$	5	One long line of output (up to 120 characters) for each iteration (no printout of the final solution).
$\geq$	10	The final solution and one long line of output for each iteration.
$\geq$	20	At each iteration, the Lagrange multipliers, the variables x, the constraint values Ax and the constraint status.
$\geq$	30	At each iteration, the diagonal elements of the upper triangular matrix T associated with the TQ factorization (4) of the working set (see Section 2 of the Mathematical Background section of this module document), and the diagonal elements of the upper triangular matrix R.

Default: `print_level` = 10.

3.2 Algorithm Choice and Tolerances

check_freq — integer

Every `check_freq` iterations, a numerical test is made to see if the current solution x satisfies the constraints in the working set. If the largest residual of the constraints in the working set is judged to be too large, the current working set is refactorized and the variables are recomputed to satisfy the constraints more accurately.

Default: `check_freq` = 50.

Constraints: `check_freq` ≥ 1.

crash_tol — real(kind=*wp*)

`crash_tol` is used in conjunction with the optional argument `cold_start` (see Section 3.2 of the procedure document for `nag_qp_sol`) in order to select an initial working set.

If `cold_start` = `.true.` (the default), the initial working set will include (if possible) bounds or general inequality constraints that lie within `crash_tol` of their bounds. In particular, a constraint of the form $a_j^T x \geq l$ will be included in the working set if $|a_j^T x - l| \leq$ `crash_tol` $\times (1 + |l|)$.

Default: `crash_tol` = 0.01.

Constraints: $0.0 \leq$ `crash_tol` ≤ 1.0.

expand_freq — integer

This forms part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems.

The strategy is to force a positive step at every iteration, at the expense of violating the constraints by a small amount. Over a period of `expand_freq` iterations, the feasibility tolerance actually used by `nag_qp_sol` (i.e., the *working* feasibility tolerance) increases from $0.5 \times$ `feas_tol` to `feas_tol` (in steps of $0.5 \times$ `feas_tol`/`expand_freq`).

At certain stages the following 'resetting procedure' is used to remove constraint infeasibilities. First, all variables whose upper or lower bounds are in the working set are moved exactly onto their bounds. A count is kept of the number of non-trivial adjustments made. If the count is positive, iterative refinement is used to give variables that satisfy the working set to (essentially) `EPSILON(1.0_`*wp*`)`. Finally, the working feasibility tolerance is reinitialized to $0.5 \times$ `feas_tol`.

If a problem requires more than `expand_freq` iterations, the resetting procedure is invoked and a new cycle of `expand_freq` iterations is started with `expand_freq` incremented by 10. (The decision to resume the feasibility phase or optimality phase is based on comparing any constraint infeasibilities with `feas_tol`.)

The resetting procedure is also invoked when `nag_qp_sol` reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any non-trivial adjustments are made, iterations are continued.

Default: `expand_freq` $= 5$.

Constraints: $0 <$ `expand_freq` $< 10^7$.

feas_tol — real(kind=*wp*)

`feas_tol` defines the maximum acceptable *absolute* violation in each constraint at a 'feasible' point. For example, if the variables and coefficients are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify `feas_tol` as 10^{-6}.

`nag_qp_sol` attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, `min_sum_of_infeas` can be used to find the minimum value of the sum. Let `Sinf` be the corresponding sum of infeasibilities. If `Sinf` is quite small, it may be appropriate to raise `feas_tol` by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

Note that a 'feasible solution' is a solution that satisfies the current constraints to within the tolerance `feas_tol`.

Default: `feas_tol` $=$ `SQRT(EPSILON(1.0_`*wp*`))`.

Constraints: `feas_tol` $\geq$ `EPSILON(1.0_`*wp*`)`.

inf_bound — real(kind=*wp*)

`inf_bound` defines the 'infinite' bound size in the definition of the problem constraints. Any upper bound greater than or equal to `inf_bound` will be regarded as $+\infty$ (and similarly any lower bound less than or equal to $-$`inf_bound` will be regarded as $-\infty$).

Default: `inf_bound` $= 10^{20}$.

Constraints: `inf_bound` > 0.0.

inf_step — real(kind=*wp*)

`inf_step` specifies the magnitude of the change in variables that will be considered a step to an unbounded solution. (Note that an unbounded solution can only occur when the Hessian is not positive definite.) If the change in x during an iteration would exceed the value of `inf_step`, the objective function is considered to be unbounded below in the feasible region.

Default: `inf_step` $= \max($`inf_bound`, $10^{20})$.

Constraints: `inf_step` $\geq$ `inf_bound`.

max_deg_free — integer

Note: max_deg_free does not apply to feasible point or linear programming problems.

It places a limit on the storage allocated for the triangular factor R of the reduced Hessian H_R (see Section 2 of the Mathematical Background section of this module document). Ideally, max_deg_free should be set slightly larger than the value of n_R expected at the solution (where $1 \leq n_R \leq$ number of variables). It need not be larger than $n_{\mathrm{N}} + 1$, where n_{N} is the number of variables that appear nonlinearly in the quadratic objective function. For many problems it can be much smaller than n_{N}.

Default: max_deg_free = number of variables if the optional argument qp_hx has been supplied, and SIZE(h,1) otherwise (see Section 3.2 of the procedure document for nag_qp_sol).

Constraints: $1 \leq$ max_deg_free $\leq$ number of variables.

feas_phase_iter_lim — integer

For a quadratic programming problem feas_phase_iter_lim specifies the maximum number of iterations allowed in the feasibility phase. For a feasible point problem it specifies the maximum number of iterations allowed before termination. For a linear programming problem the maximum number of iterations allowed before termination is taken as max(feas_phase_iter_lim, opt_phase_iter_lim).

If you wish to check that a call to nag_qp_sol is correct before attempting to solve the problem in full then feas_phase_iter_lim may be set to 0. No iterations will be performed but the initialization stages prior to the first iteration will be processed and a listing of parameter settings output if list = .true. (the default).

Default: feas_phase_iter_lim = max(50, 5 × (number of variables + number of linear constraints)).

Constraints: feas_phase_iter_lim ≥ 0.

opt_phase_iter_lim — integer

Note: opt_phase_iter_lim does not apply to feasible point problems.

For a linear programming problem the maximum number of iterations allowed before termination is taken as max(feas_phase_iter_lim, opt_phase_iter_lim). For a quadratic programming problem it specifies the maximum number of iterations allowed in the optimality phase.

Default: opt_phase_iter_lim = max(50, 5 × (number of variables + number of linear constraints)).

Constraints: opt_phase_iter_lim ≥ 0.

rank_tol — real(kind=*wp*)

Note: rank_tol does not apply to feasible point or linear programming problems.

It enables you to control the condition number of the triangular factor R (see Section 4 of the Mathematical Background section of this module document). If $\rho_i = \max\{|R_{11}|, |R_{22}|, \ldots, |R_{ii}|\}$, the dimension of R is defined to be the smallest index i such that $|R_{i+1,i+1}| \leq \sqrt{\texttt{rank\_tol}} \times |\rho_{i+1}|$.

Default: rank_tol = 100 × EPSILON(1.0_*wp*).

Constraints: rank_tol > 0.0.

min_sum_of_infeas — logical

It enables you to control whether or not nag_qp_sol will calculate a point that minimizes the constraint violations when no feasible point exists.

If min_sum_of_infeas = .false., termination will occur as soon as it is evident that no feasible point exists for the constraints. The final point will generally not be the point at which the sum of infeasibilities is minimized.

If min_sum_of_infeas = .true., termination will occur either after the sum of infeasibilities has been minimized or the iteration limit has been reached, whichever occurs first.

Default: min_sum_of_infeas = .false..

Example 1: Feasible point problem

To find a feasible point for the bounds

$$\begin{array}{rcrcl} -0.01 & \leq & x_1 & \leq & 0.01 \\ -0.10 & \leq & x_2 & \leq & 0.15 \\ -0.01 & \leq & x_3 & \leq & 0.03 \\ -0.04 & \leq & x_4 & \leq & 0.02 \\ -0.10 & \leq & x_5 & \leq & 0.05 \\ -0.01 & \leq & x_6 & & \\ -0.01 & \leq & x_7 & & \end{array}$$

and the linear constraints

$$\begin{array}{rl} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = -0.1300 \\ & 0.15x_1 + 0.04x_2 + 0.02x_3 + 0.04x_4 + 0.02x_5 + 0.01x_6 + 0.03x_7 \leq -0.0049 \\ & 0.03x_1 + 0.05x_2 + 0.08x_3 + 0.02x_4 + 0.06x_5 + 0.01x_6 \leq -0.0064 \\ & 0.02x_1 + 0.04x_2 + 0.01x_3 + 0.02x_4 + 0.02x_5 \leq -0.0037 \\ & 0.02x_1 + 0.03x_2 + 0.01x_5 \leq -0.0012 \\ -0.0992 \leq & 0.70x_1 + 0.75x_2 + 0.80x_3 + 0.75x_4 + 0.80x_5 + 0.97x_6 \\ -0.0030 \leq & 0.02x_1 + 0.06x_2 + 0.08x_3 + 0.12x_4 + 0.02x_5 + 0.01x_6 + 0.97x_7 \leq 0.0020 \end{array}$$

The initial point is

$$x^{(0)} = (-0.01,\ -0.03,\ 0.0,\ -0.01,\ -0.1,\ 0.02,\ 0.01)^T.$$

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_qp_ex01

  ! Example Program Text for nag_qp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_qp, ONLY : nag_qp_sol
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  REAL (wp) :: obj_f
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), ax_lower(:), ax_upper(:), x(:), &
   x_lower(:), x_upper(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_qp_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file

  ! Read number of linear constraints (m) and variables (n)
  READ (nag_std_in,*) m, n

  ALLOCATE (a(m,n),ax_lower(m),ax_upper(m),x(n),x_lower(n), &
   x_upper(n))                 ! Allocate storage
```

```
      ! Read in problem data
      READ (nag_std_in,*) (a(i,:),i=1,m)
      READ (nag_std_in,*) x_lower
      READ (nag_std_in,*) ax_lower
      READ (nag_std_in,*) x_upper
      READ (nag_std_in,*) ax_upper
      READ (nag_std_in,*) x

      ! Solve the problem

      CALL nag_qp_sol(x,obj_f,a=a,ax_lower=ax_lower,ax_upper=ax_upper, &
       x_lower=x_lower,x_upper=x_upper)

      DEALLOCATE (a,ax_lower,ax_upper,x,x_lower,x_upper) ! Deallocate storage

    END PROGRAM nag_qp_ex01
```

2 Program Data

```
Example Program Data for nag_qp_ex01
 7  7                                                  :Values of m and n
  1.00   1.00   1.00   1.00   1.00   1.00   1.00
  0.15   0.04   0.02   0.04   0.02   0.01   0.03
  0.03   0.05   0.08   0.02   0.06   0.01   0.00
  0.02   0.04   0.01   0.02   0.02   0.00   0.00
  0.02   0.03   0.00   0.00   0.01   0.00   0.00
  0.70   0.75   0.80   0.75   0.80   0.97   0.00
  0.02   0.06   0.08   0.12   0.02   0.01   0.97   :End of a
 -0.01  -0.10    -0.01    -0.04    -0.10    -0.01     -0.01    :End of x_lower
 -0.13  -1.0e+25 -1.0e+25 -1.0e+25 -1.0e+25 -9.92e-02 -3.0e-03 :End of ax_lower
  0.01   0.15     0.03     0.02     0.05     1.0e+25   1.0e+25 :End of x_upper
 -0.13  -4.9e-03 -6.4e-03 -3.7e-03 -1.2e-03  1.0e+25   2.0e-03 :End of ax_upper
 -0.01  -0.03     0.00    -0.01    -0.10     0.02      0.01    :End of x
```

3 Program Results

```
 Example Program Results for nag_qp_ex01

 Parameters
 ----------

 prob_type.............         F     (feasible point problem)

 linear constraints.....         7     variables.............         7

 list...................    .true.     lt80_char.............    .true.
 print_level............        10

 feas_tol..............  1.49E-08     crash_tol.............  1.00E-02

 inf_bound.............  1.00E+20     cold_start............    .true.
 inf_step..............  1.00E+20     eps (machine precision)  2.22E-16

 check_freq............        50     feas_phase_iter_lim....        70
 expand_freq...........         5     unit..................         6
 min_sum_of_infeas.....   .false.


  Itn     Step Ninf Sinf/Objective  Norm Gz
```

```
   0  0.0E+00    3   1.038000E-01  0.0E+00
   1  4.1E-02    1   3.000000E-02  0.0E+00
   2  4.2E-02    0   0.000000E+00  0.0E+00


 Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 V   1  A LL  -1.000000E-02 -1.000000E-02  1.000000E-02       .         .
 V   2    FR  -1.500000E-02 -0.100000      0.150000           .      8.5000E-02
 V   3  A LL  -1.000000E-02 -1.000000E-02  3.000000E-02       .         .
 V   4  A LL  -4.000000E-02 -4.000000E-02  2.000000E-02       .         .
 V   5  A LL  -0.100000     -0.100000      5.000000E-02       .         .
 V   6    FR   3.819588E-02 -1.000000E-02      None           .      4.8196E-02
 V   7    FR   6.804124E-03 -1.000000E-02      None           .      1.6804E-02


 L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 L   1  A EQ  -0.130000     -0.130000     -0.130000           .         .
 L   2    FR  -5.313918E-03      None     -4.900000E-03       .      4.1392E-04
 L   3    FR  -8.268041E-03      None     -6.400000E-03       .      1.8680E-03
 L   4  A UL  -3.700000E-03      None     -3.700000E-03       .     -4.3368E-19
 L   5    FR  -1.650000E-03      None     -1.200000E-03       .      4.5000E-04
 L   6  A LL  -9.920000E-02 -9.920000E-02      None           .      2.7756E-17
 L   7    FR  -1.718041E-03 -3.000000E-03  2.000000E-03       .      1.2820E-03

 Exit nag_qp_sol - Feasible point found.

 Exit from nag_qp_sol after     2 iterations.
```

Example 2: Linear programming problem

To minimize the linear function

$$-0.02x_1 - 0.2x_2 - 0.2x_3 - 0.2x_4 - 0.2x_5 + 0.04x_6 + 0.04x_7$$

using the set of constraints given in Example 1.

The initial point, which is infeasible, is

$$x^{(0)} = (-0.01,\ -0.03,\ 0.0,\ -0.01,\ -0.1,\ 0.02,\ 0.01)^T.$$

The optimal solution (to five figures) is

$$x^* = (-0.01,\ -0.1,\ 0.03,\ 0.02,\ -0.067485,\ -0.0022801,\ -0.00023453)^T.$$

Four bound constraints and three linear constraints are active at the solution.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_qp_ex02

  ! Example Program Text for nag_qp
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_qp, ONLY : nag_qp_sol, nag_qp_cntrl_init, &
   nag_qp_cntrl_wp => nag_qp_cntrl_dp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  REAL (wp) :: obj_f
  TYPE (nag_qp_cntrl_wp) :: control
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), ax_lower(:), ax_upper(:), c(:), x(:), &
   x_lower(:), x_upper(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_qp_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file

  ! Read number of linear constraints (m) and variables (n)
  READ (nag_std_in,*) m, n

  ALLOCATE (a(m,n),ax_lower(m),ax_upper(m),c(n),x(n),x_lower(n), &
   x_upper(n))                 ! Allocate storage

  ! Read in problem data
  READ (nag_std_in,*) c
  READ (nag_std_in,*) (a(i,:),i=1,m)
  READ (nag_std_in,*) x_lower
  READ (nag_std_in,*) ax_lower
  READ (nag_std_in,*) x_upper
```

```
      READ (nag_std_in,*) ax_upper
      READ (nag_std_in,*) x

      ! initialize control structure and set required control parameters

      CALL nag_qp_cntrl_init(control)

      control%print_level = 1

      ! Solve the problem

      CALL nag_qp_sol(x,obj_f,prob_type='L',a=a,ax_lower=ax_lower, &
       ax_upper=ax_upper,c=c,x_lower=x_lower,x_upper=x_upper,control=control)

      DEALLOCATE (a,ax_lower,ax_upper,c,x,x_lower,x_upper) ! Deallocate storage

    END PROGRAM nag_qp_ex02
```

2 Program Data

```
Example Program Data for nag_qp_ex02
 7  7                                                  :Values of m and n
 -0.02  -0.20  -0.20  -0.20  -0.20   0.04   0.04   :End of c
  1.00   1.00   1.00   1.00   1.00   1.00   1.00
  0.15   0.04   0.02   0.04   0.02   0.01   0.03
  0.03   0.05   0.08   0.02   0.06   0.01   0.00
  0.02   0.04   0.01   0.02   0.02   0.00   0.00
  0.02   0.03   0.00   0.00   0.01   0.00   0.00
  0.70   0.75   0.80   0.75   0.80   0.97   0.00
  0.02   0.06   0.08   0.12   0.02   0.01   0.97   :End of a
 -0.01 -0.10    -0.01    -0.04    -0.10    -0.01     -0.01    :End of x_lower
 -0.13 -1.0e+25 -1.0e+25 -1.0e+25 -1.0e+25 -9.92e-02 -3.0e-03 :End of ax_lower
  0.01  0.15     0.03     0.02     0.05     1.0e+25   1.0e+25 :End of x_upper
 -0.13 -4.9e-03 -6.4e-03 -3.7e-03 -1.2e-03  1.0e+25   2.0e-03 :End of ax_upper
 -0.01 -0.03     0.00    -0.01    -0.10     0.02      0.01    :End of x
```

3 Program Results

```
 Example Program Results for nag_qp_ex02

 Parameters
 ----------

 prob_type.............         L     (linear programming problem)

 linear constraints.....         7     variables.............         7

 list...................    .true.     lt80_char.............    .true.
 print_level............         1

 feas_tol..............  1.49E-08     crash_tol.............  1.00E-02

 inf_bound.............  1.00E+20     cold_start............    .true.
 inf_step..............  1.00E+20     eps (machine precision)  2.22E-16

 check_freq............        50     feas_phase_iter_lim....        70
 min_sum_of_infeas.....   .false.     opt_phase_iter_lim.....        70
 expand_freq...........         5     unit...................         6


 Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack
```

```
V   1    LL  -1.000000E-02 -1.000000E-02  1.000000E-02  0.3301         .
V   2    LL  -0.100000     -0.100000      0.150000      1.4384E-02     .
V   3    UL   3.000000E-02 -1.000000E-02  3.000000E-02 -9.0997E-02     .
V   4    UL   2.000000E-02 -4.000000E-02  2.000000E-02 -7.6612E-02     .
V   5    FR  -6.748534E-02 -0.100000      5.000000E-02       .      3.2515E-02
V   6    FR  -2.280130E-03 -1.000000E-02      None           .      7.7199E-03
V   7    FR  -2.345277E-04 -1.000000E-02      None           .      9.7655E-03


L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

L   1    EQ  -0.130000     -0.130000     -0.130000      -1.431     -5.5511E-17
L   2    FR  -5.479544E-03      None     -4.900000E-03       .      5.7954E-04
L   3    FR  -6.571922E-03      None     -6.400000E-03       .      1.7192E-04
L   4    FR  -4.849707E-03      None     -3.700000E-03       .      1.1497E-03
L   5    FR  -3.874853E-03      None     -1.200000E-03       .      2.6749E-03
L   6    LL  -9.920000E-02 -9.920000E-02      None       1.501      1.3878E-17
L   7    LL  -3.000000E-03 -3.000000E-03  2.000000E-03   1.517      2.6021E-18

Exit nag_qp_sol - Optimal LP solution.

Final LP objective value =   0.2359648E-01

Exit from nag_qp_sol after     7 iterations.
```

Example 3: Quadratic programming problem (explicit H)

To minimize the quadratic function

$$f(x) = c^T x + \tfrac{1}{2} x^T H x,$$

where

$$c = (-0.02,\ -0.2,\ -0.2,\ -0.2,\ -0.2,\ 0.04,\ 0.04)^T$$

$$H = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 & -2 & -2 \end{pmatrix}$$

using the set of constraints given in Example 1.

The initial point, which is infeasible, is

$$x^{(0)} = (-0.01,\ -0.03,\ 0.0,\ -0.01,\ -0.1,\ 0.02,\ 0.01)^T.$$

The optimal solution (to five figures) is

$$x^* = (-0.01,\ -0.069865,\ 0.018259,\ -0.24261,\ -0.62006,\ 0.013805,\ 0.0040665)^T.$$

One bound constraint and four linear constraints are active at the solution.

This example uses the optional argument `h` (see Section 3.2 of the procedure document for `nag_qp_sol`) to store H explicitly.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_qp_ex03

  ! Example Program Text for nag_qp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_qp, ONLY : nag_qp_sol
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, n
  REAL (wp) :: obj_f
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), ax_lower(:), ax_upper(:), c(:), &
   h(:,:), x(:), x_lower(:), x_upper(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_qp_ex03'
```

```
      READ (nag_std_in,*)          ! Skip heading in data file

      ! Read number of linear constraints (m) and variables (n)
      READ (nag_std_in,*) m, n

      ALLOCATE (a(m,n),ax_lower(m),ax_upper(m),c(n),h(n,n),x(n),x_lower(n), &
       x_upper(n))                 ! Allocate storage

      ! Read in problem data
      READ (nag_std_in,*) c
      READ (nag_std_in,*) (a(i,:),i=1,m)
      READ (nag_std_in,*) x_lower
      READ (nag_std_in,*) ax_lower
      READ (nag_std_in,*) x_upper
      READ (nag_std_in,*) ax_upper
      READ (nag_std_in,*) x
      READ (nag_std_in,*) (h(i,:),i=1,n)

      ! Solve the problem

      CALL nag_qp_sol(x,obj_f,prob_type='S',h=h,a=a,ax_lower=ax_lower, &
       ax_upper=ax_upper,c=c,x_lower=x_lower,x_upper=x_upper)

      DEALLOCATE (a,ax_lower,ax_upper,c,h,x,x_lower, &
       x_upper)                    ! Deallocate storage

    END PROGRAM nag_qp_ex03
```

2 Program Data

```
Example Program Data for nag_qp_ex03
 7  7                                                    :Values of m and n
 -0.02  -0.20  -0.20  -0.20  -0.20   0.04   0.04   :End of c
  1.00   1.00   1.00   1.00   1.00   1.00   1.00
  0.15   0.04   0.02   0.04   0.02   0.01   0.03
  0.03   0.05   0.08   0.02   0.06   0.01   0.00
  0.02   0.04   0.01   0.02   0.02   0.00   0.00
  0.02   0.03   0.00   0.00   0.01   0.00   0.00
  0.70   0.75   0.80   0.75   0.80   0.97   0.00
  0.02   0.06   0.08   0.12   0.02   0.01   0.97   :End of a
 -0.01 -0.10    -0.01    -0.04    -0.10    -0.01     -0.01    :End of x_lower
 -0.13 -1.0e+25 -1.0e+25 -1.0e+25 -1.0e+25 -9.92e-02 -3.0e-03 :End of ax_lower
  0.01  0.15     0.03     0.02     0.05     1.0e+25   1.0e+25 :End of x_upper
 -0.13 -4.9e-03 -6.4e-03 -3.7e-03 -1.2e-03  1.0e+25   2.0e-03 :End of ax_upper
 -0.01  -0.03   0.00  -0.01  -0.10   0.02   0.01   :End of x
  2.00   0.00   0.00   0.00   0.00   0.00   0.00
  0.00   2.00   0.00   0.00   0.00   0.00   0.00
  0.00   0.00   2.00   2.00   0.00   0.00   0.00
  0.00   0.00   2.00   2.00   0.00   0.00   0.00
  0.00   0.00   0.00   0.00   2.00   0.00   0.00
  0.00   0.00   0.00   0.00   0.00  -2.00  -2.00
  0.00   0.00   0.00   0.00   0.00  -2.00  -2.00   :End of h
```

3 Program Results

```
Example Program Results for nag_qp_ex03

Parameters
----------

prob_type.............         S     (symmetric QP problem)

linear constraints.....         7     variables..............         7

list...................    .true.     lt80_char..............    .true.
print_level............        10

feas_tol...............  1.49E-08     crash_tol..............  1.00E-02

inf_bound..............  1.00E+20     cold_start.............    .true.
inf_step...............  1.00E+20     eps (machine precision)  2.22E-16

check_freq.............        50     feas_phase_iter_lim....        70
expand_freq............         5     opt_phase_iter_lim.....        70

max_deg_free...........         7     rank_tol...............  2.22E-14
unit...................         6     min_sum_of_infeas......   .false.


 Itn     Step Ninf Sinf/Objective  Norm Gz
   0  0.0E+00    3   1.038000E-01  0.0E+00
   1  4.1E-02    1   3.000000E-02  0.0E+00
   2  4.2E-02    0   0.000000E+00  0.0E+00
Itn     2 -- Feasible point found.
   2  0.0E+00    0   4.580000E-02  0.0E+00
   3  1.3E-01    0   4.161596E-02  0.0E+00
   4  1.0E+00    0   3.936227E-02  0.0E+00
   5  4.1E-01    0   3.758935E-02  1.2E-02
   6  1.0E+00    0   3.755377E-02  2.8E-17
   7  1.0E+00    0   3.703165E-02  4.3E-17


Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

V   1    LL  -1.000000E-02 -1.000000E-02  1.000000E-02  0.4700         .
V   2    FR  -6.986465E-02 -0.100000      0.150000            .      3.0135E-02
V   3    FR   1.825915E-02 -1.000000E-02  3.000000E-02        .      1.1741E-02
V   4    FR  -2.426081E-02 -4.000000E-02  2.000000E-02        .      1.5739E-02
V   5    FR  -6.200564E-02 -0.100000      5.000000E-02        .      3.7994E-02
V   6    FR   1.380544E-02 -1.000000E-02      None            .      2.3805E-02
V   7    FR   4.066496E-03 -1.000000E-02      None            .      1.4066E-02


L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

L   1    EQ  -0.130000     -0.130000     -0.130000      -1.908     -5.5511E-17
L   2    FR  -5.879898E-03      None     -4.900000E-03        .      9.7990E-04
L   3    UL  -6.400000E-03      None     -6.400000E-03 -0.3144      1.7347E-18
L   4    FR  -4.537323E-03      None     -3.700000E-03        .      8.3732E-04
L   5    FR  -2.915996E-03      None     -1.200000E-03        .      1.7160E-03
L   6    LL  -9.920000E-02 -9.920000E-02      None        1.955         .
L   7    LL  -3.000000E-03 -3.000000E-03  2.000000E-03    1.972     -6.0715E-18

Exit nag_qp_sol - Optimal QP solution.

Final QP objective value =   0.3703165E-01
```

```
Exit from nag_qp_sol after     7 iterations.
```

Example 4: Quadratic programming problem (implicit H)

This example solves the same problem as Example 3, but uses the optional argument `qp_hx` (see Section 3.2 of the procedure document for `nag_qp_sol`) to avoid referencing H explicitly.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE qp_ex04_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  SUBROUTINE qp_hx(col,x,hx)

    ! Procedure to evaluate h*x for some matrix h that defines the
    ! hessian of the required qp problem. In this version of qp_hx,
    ! h is implicit.

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Scalar Arguments ..
    INTEGER, INTENT (IN) :: col
    ! .. Array Arguments ..
    REAL (wp), INTENT (INOUT) :: hx(:)
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Executable Statements ..

    hx(1) = 2.0_wp*x(1)
    hx(2) = 2.0_wp*x(2)
    hx(3) = 2.0_wp*(x(3)+x(4))
    hx(4) = hx(3)
    hx(5) = 2.0_wp*x(5)
    hx(6) = -2.0_wp*(x(6)+x(7))
    hx(7) = hx(6)

  END SUBROUTINE qp_hx

END MODULE qp_ex04_mod

PROGRAM nag_qp_ex04

  ! Example Program Text for nag_qp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_qp, ONLY : nag_qp_sol
  USE qp_ex04_mod, ONLY : qp_hx, wp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Local Scalars ..
  INTEGER :: i, m, n
```

```
      REAL (wp) :: obj_f
      ! .. Local Arrays ..
      REAL (wp), ALLOCATABLE :: a(:,:), ax_lower(:), ax_upper(:), c(:), x(:), &
       x_lower(:), x_upper(:)
      ! .. Executable Statements ..

      WRITE (nag_std_out,*) 'Example Program Results for nag_qp_ex04'

      READ (nag_std_in,*)          ! Skip heading in data file

      ! Read number of linear constraints (m) and variables (n)
      READ (nag_std_in,*) m, n

      ALLOCATE (a(m,n),ax_lower(m),ax_upper(m),c(n),x(n),x_lower(n), &
       x_upper(n))                 ! Allocate storage

      ! Read in problem data
      READ (nag_std_in,*) c
      READ (nag_std_in,*) (a(i,:),i=1,m)
      READ (nag_std_in,*) x_lower
      READ (nag_std_in,*) ax_lower
      READ (nag_std_in,*) x_upper
      READ (nag_std_in,*) ax_upper
      READ (nag_std_in,*) x

      ! Solve the problem

      CALL nag_qp_sol(x,obj_f,prob_type='S',qp_hx=qp_hx,a=a,ax_lower=ax_lower, &
       ax_upper=ax_upper,c=c,x_lower=x_lower,x_upper=x_upper)

      DEALLOCATE (a,ax_lower,ax_upper,c,x,x_lower, &
       x_upper)                    ! Deallocate storage

    END PROGRAM nag_qp_ex04
```

2 Program Data

```
Example Program Data for nag_qp_ex04
 7  7                                                :Values of m and n
 -0.02  -0.20  -0.20  -0.20  -0.20   0.04   0.04   :End of c
  1.00   1.00   1.00   1.00   1.00   1.00   1.00
  0.15   0.04   0.02   0.04   0.02   0.01   0.03
  0.03   0.05   0.08   0.02   0.06   0.01   0.00
  0.02   0.04   0.01   0.02   0.02   0.00   0.00
  0.02   0.03   0.00   0.00   0.01   0.00   0.00
  0.70   0.75   0.80   0.75   0.80   0.97   0.00
  0.02   0.06   0.08   0.12   0.02   0.01   0.97   :End of a
 -0.01 -0.10    -0.01    -0.04    -0.10    -0.01     -0.01    :End of x_lower
 -0.13 -1.0e+25 -1.0e+25 -1.0e+25 -1.0e+25 -9.92e-02 -3.0e-03 :End of ax_lower
  0.01  0.15     0.03     0.02     0.05     1.0e+25   1.0e+25 :End of x_upper
 -0.13 -4.9e-03 -6.4e-03 -3.7e-03 -1.2e-03  1.0e+25   2.0e-03 :End of ax_upper
 -0.01 -0.03     0.00    -0.01    -0.10     0.02      0.01    :End of x
```

3 Program Results

```
 Example Program Results for nag_qp_ex04

 Parameters
 ----------

 prob_type.............         S     (symmetric QP problem)
```

```
linear constraints.....         7     variables.............         7

list...................    .true.     lt80_char.............    .true.
print_level............        10

feas_tol...............  1.49E-08     crash_tol.............  1.00E-02

inf_bound..............  1.00E+20     cold_start............    .true.
inf_step...............  1.00E+20     eps (machine precision)  2.22E-16

check_freq.............        50     feas_phase_iter_lim....        70
expand_freq............         5     opt_phase_iter_lim.....        70

max_deg_free...........         7     rank_tol...............  2.22E-14
unit...................         6     min_sum_of_infeas......   .false.


 Itn     Step Ninf Sinf/Objective  Norm Gz
   0  0.0E+00    3   1.038000E-01  0.0E+00
   1  4.1E-02    1   3.000000E-02  0.0E+00
   2  4.2E-02    0   0.000000E+00  0.0E+00
Itn     2 -- Feasible point found.
   2  0.0E+00    0   4.580000E-02  0.0E+00
   3  1.3E-01    0   4.161596E-02  0.0E+00
   4  1.0E+00    0   3.936227E-02  0.0E+00
   5  4.1E-01    0   3.758935E-02  1.2E-02
   6  1.0E+00    0   3.755377E-02  2.8E-17
   7  1.0E+00    0   3.703165E-02  4.3E-17


Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

V   1    LL  -1.000000E-02 -1.000000E-02  1.000000E-02  0.4700         .
V   2    FR  -6.986465E-02 -0.100000      0.150000            .      3.0135E-02
V   3    FR   1.825915E-02 -1.000000E-02  3.000000E-02        .      1.1741E-02
V   4    FR  -2.426081E-02 -4.000000E-02  2.000000E-02        .      1.5739E-02
V   5    FR  -6.200564E-02 -0.100000      5.000000E-02        .      3.7994E-02
V   6    FR   1.380544E-02 -1.000000E-02      None            .      2.3805E-02
V   7    FR   4.066496E-03 -1.000000E-02      None            .      1.4066E-02


L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

L   1    EQ  -0.130000     -0.130000     -0.130000      -1.908     -5.5511E-17
L   2    FR  -5.879898E-03      None     -4.900000E-03       .      9.7990E-04
L   3    UL  -6.400000E-03      None     -6.400000E-03 -0.3144      1.7347E-18
L   4    FR  -4.537323E-03      None     -3.700000E-03       .      8.3732E-04
L   5    FR  -2.915996E-03      None     -1.200000E-03       .      1.7160E-03
L   6    LL  -9.920000E-02 -9.920000E-02      None       1.955         .
L   7    LL  -3.000000E-03 -3.000000E-03  2.000000E-03   1.972     -6.0715E-18

Exit nag_qp_sol - Optimal QP solution.

Final QP objective value =   0.3703165E-01

Exit from nag_qp_sol after     7 iterations.
```

Mathematical Background

1 Overview

`nag_qp_sol` is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian. The method is based on that of Gill and Murray [2] and is described in detail by Gill *et al.* [5]. Here we briefly summarize the main features of the method. Where possible, explicit reference is made to the names of variables that are arguments of `nag_qp_sol` or appear in the printed output. `nag_qp_sol` has two phases: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same procedures. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function. The feasibility phase does *not* perform the standard simplex method (i.e., it does not necessarily find a vertex), except for problems of type L when $n_{\mathrm{L}} \leq n$. Once any iterate is feasible, all subsequent iterates remain feasible.

`nag_qp_sol` has been designed to be efficient when used to solve a *sequence* of related problems, for example within a sequential quadratic programming method for nonlinearly constrained optimization. In particular, you may specify the initial working set (the indices of the constraints believed to be satisfied exactly at the solution); see the discussion of the optional argument `cold_start` in Section 3.2 of the procedure document for `nag_qp_sol`.

In general, an iterative process is required to solve a quadratic program. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{2}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*.

At each point x, a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the tolerance defined by `control%feas_tol`; see the type definition for `nag_qp_cntrl_wp`). The working set is the current prediction of the constraints that hold with equality at a solution of a linearly constrained QP problem. The search direction is constructed so that the constraints in the working set remain *unaltered* for any value of the step length. For a bound constraint in the working set, this property is achieved by setting the corresponding element of the search direction to zero. Thus the associated variable is *fixed*, and specification of the working set induces a partition of x into *fixed* and *free* variables. During a given iteration the fixed variables are effectively removed from the problem; since the relevant elements of the search direction are zero, the columns of A corresponding to fixed variables may be ignored.

The constraints involving A are called the *general* constraints. Let n_{W} denote the number of general constraints in the working set and let n_{FX} denote the number of variables fixed at one of their bounds (n_{W} and n_{FX} are the quantities `Lin` and `Bnd` in the printed output; see Section 7 of the procedure document for `nag_qp_sol`). Similarly, let n_{FR} ($n_{\mathrm{FR}} = n - n_{\mathrm{FX}}$) denote the number of free variables. At every iteration, *the variables are reordered so that the last n_{FX} variables are fixed*, with all other relevant vectors and matrices ordered accordingly.

2 Definition of the Search Direction

Let A_{FR} denote the n_{W} by n_{FR} sub-matrix of general constraints in the working set corresponding to the free variables, and let p_{FR} denote the search direction with respect to the free variables only. The general constraints in the working set will be unaltered by any move along p if

$$A_{\mathrm{FR}} p_{\mathrm{FR}} = 0. \tag{3}$$

In order to compute p_{FR}, the *TQ factorization* of A_{FR} is used:

$$A_{\mathrm{FR}} Q_{\mathrm{FR}} = (0 \;\; T), \tag{4}$$

where T is a non-singular n_{W} by n_{W} upper triangular matrix (i.e., $t_{ij} = 0$ if $i > j$), and the non-singular n_{FR} by n_{FR} matrix Q_{FR} is the product of orthogonal transformations (see Gill *et al.* [3]). If the columns of Q_{FR} are partitioned so that

$$Q_{\mathrm{FR}} = (Z \;\; Y),$$

where Y is n_{FR} by n_{W}, then the n_Z ($n_Z = n_{\mathrm{FR}} - n_{\mathrm{W}}$) columns of Z form a basis for the null space of A_{FR}. Let n_R be an integer such that $0 \leq n_R \leq n_Z$, and let Z_R denote a matrix whose n_R columns are a subset of the columns of Z. (The integer n_R is the quantity `Zr` in the printed output; see Section 7 of the procedure document for `nag_qp_sol`. In many cases, Z_R will include *all* the columns of Z.) The direction p_{FR} will satisfy (3) if

$$p_{\mathrm{FR}} = Z_R p_R, \tag{5}$$

where p_R is any n_R-vector.

Let Q denote the n by n matrix

$$Q = \begin{pmatrix} Q_{\mathrm{FR}} & \\ & I_{\mathrm{FX}} \end{pmatrix}$$

where I_{FX} is the identity matrix of order n_{FX}. Let H_Q and g_Q denote the n by n *transformed Hessian* and *transformed gradient*

$$H_Q = Q^T H Q \;\; \text{and} \;\; g_Q = Q^T(c + Hx)$$

and let the matrix of first n_R rows and columns of H_Q be denoted by H_R and the vector of the first n_R elements of g_Q be denoted by g_R. The quantities H_R and g_R are known as the *reduced Hessian* and *reduced gradient* of $f(x)$, respectively. Roughly speaking, g_R and H_R describe the first and second derivatives of an *unconstrained* problem for the calculation of p_R.

At each iteration, a triangular factorization of H_R is available. If H_R is positive definite, $H_R = R^T R$, where R is the upper triangular Cholesky factor of H_R. If H_R is not positive definite, $H_R = R^T D R$, where $D = \mathrm{diag}(1, 1, \ldots, 1, \mu)$, with $\mu \leq 0$.

The computation is arranged so that the reduced-gradient vector is a multiple of e_R, a vector of all zeros except in the last (i.e., n_Rth) position. This allows the vector p_R in (5) to be computed from a single back-substitution

$$R p_R = \gamma e_R, \tag{6}$$

where γ is a scalar that depends on whether or not the reduced Hessian is positive definite at x. In the positive-definite case, $x + p$ is the minimizer of the objective function subject to the constraints (bounds and general) in the working set treated as equalities. If H_R is not positive definite, p_R satisfies the conditions

$$p_R^T H_R p_R < 0 \;\; \text{and} \;\; g_R^T p_R \leq 0,$$

which allow the objective function to be reduced by any positive step of the form $x + \alpha p$.

3 The Main Iteration

If the reduced gradient is zero, x is a constrained stationary point in the sub-space defined by Z. During the feasibility phase, the reduced gradient will usually be zero only at a vertex (although it may be zero at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero reduced gradient implies that x minimizes the quadratic objective when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ_C and λ_B for the general and bound constraints are defined from the equations

$$A_{\mathrm{FR}}^T \lambda_C = g_{\mathrm{FR}} \;\; \text{and} \;\; \lambda_B = g_{\mathrm{FX}} - A_{\mathrm{FX}}^T \lambda_C. \tag{7}$$

Given a positive constant δ of the order of `EPSILON(1.0_wp)`, a Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \delta$ when the associated constraint

is at its *upper bound*, or if $\lambda_j \geq -\delta$ when the associated constraint is at its *lower bound*. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is non-zero, there is no feasible point, and you can force `nag_qp_sol` to continue until the minimum value of the sum of infeasibilities has been found; see the discussion of `control%min_sum_of_infeas` in the type definition for `nag_qp_cntrl_wp`. At such a point, the Lagrange multiplier λ_j corresponding to an inequality constraint in the working set will be such that $-(1+\delta) \leq \lambda_j \leq \delta$ when the associated constraint is at its *upper bound*, and $-\delta \leq \lambda_j \leq (1+\delta)$ when the associated constraint is at its *lower bound*. Lagrange multipliers for equality constraints will satisfy $\mid \lambda_j \mid \leq (1+\delta)$.

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the non-zero elements of the search direction p are given by $Z_R p_R$ (see (5) and (6)). The choice of step length is influenced by the need to maintain feasibility with respect to the satisfied constraints. If H_R is positive definite and $x+p$ is feasible, α will be taken as unity. In this case, the reduced gradient at $\bar{x}$ will be zero, and Lagrange multipliers are computed. Otherwise, α is set to α_{M}, the step to the 'nearest' constraint (with index `Jadd`; see Section 7 of the procedure document for `nag_qp_sol`), which is added to the working set at the next iteration.

Each change in the working set leads to a simple change to A_{FR}: if the status of a general constraint changes, a *row* of A_{FR} is altered; if a bound constraint enters or leaves the working set, a *column* of A_{FR} changes. Explicit representations are recurred of the matrices T, Q_{FR} and R; and of vectors $Q^T g$ and $Q^T c$. The triangular factor R associated with the reduced Hessian is only updated during the optimality phase.

One of the most important features of `nag_qp_sol` is its control of the working set, whose nearness to linear dependence is estimated by the ratio of the largest to smallest diagonal elements of the TQ factor T (the printed value `Cond T`; see Section 7 of the procedure document for `nag_qp_sol`). In constructing the initial working set, constraints are excluded that would result in a large ratio.

`nag_qp_sol` includes a rigorous procedure that prevents the possibility of cycling at a point where the active constraints are nearly linearly dependent (see Gill *et al.* [4]). The main feature of the anti-cycling procedure is that the feasibility tolerance is increased slightly at the start of every iteration. This not only allows a positive step to be taken at every iteration, but also provides, whenever possible, a *choice* of constraints to be added to the working set. Let α_{M} denote the maximum step at which $x + \alpha_{\text{M}} p$ does not violate any constraint by more than its feasibility tolerance. All constraints at a distance α ($\alpha \leq \alpha_{\text{M}}$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the largest angle with the search direction is added to the working set.

4 Choosing the Initial Working Set

At the start of the optimality phase, a positive definite H_R can be defined if enough constraints are included in the initial working set. (The matrix with no rows and columns is positive definite by definition, corresponding to the case when A_{FR} contains n_{FR} constraints.) The idea is to include as many general constraints as necessary to ensure that the reduced Hessian is positive definite.

Let H_Z denote the matrix of the first n_Z rows and columns of the matrix $H_Q = Q^T H Q$ at the beginning of the optimality phase. A partial Cholesky factorization is used to find an upper triangular matrix R that is the factor of the largest positive-definite leading sub-matrix of H_Z. The use of interchanges during the factorization of H_Z tends to maximize the dimension of R. (The condition of R may be controlled using `control%rank_tol`; see the type definition for `nag_qp_cntrl_wp`.) Let Z_R denote the columns of Z corresponding to R, and let Z be partitioned as $Z = (Z_R \; Z_A)$. A working set for which Z_R defines the null space can be obtained by including *the rows of* Z_A^T as 'artificial constraints'. Minimization of the objective function then proceeds within the sub-space defined by Z_R, as described in Section 2.

The artificially augmented working set is given by

$$\bar{A}_{\text{FR}} = \begin{pmatrix} Z_A^T \\ A_{\text{FR}} \end{pmatrix}, \tag{8}$$

so that $p_{\rm FR}$ will satisfy $A_{\rm FR}p_{\rm FR} = 0$ *and* $Z_A^T p_{\rm FR} = 0$. By definition of the TQ factorization, $\bar{A}_{\rm FR}$ *automatically* satisfies the following:

$$\bar{A}_{\rm FR}Q_{\rm FR} = \begin{pmatrix} Z_A^T \\ A_{\rm FR} \end{pmatrix} Q_{\rm FR} = \begin{pmatrix} Z_A^T \\ A_{\rm FR} \end{pmatrix} (Z_R \;\; Z_A \;\; Y) = (0 \;\; \bar{T}),$$

where

$$\bar{T} = \begin{pmatrix} I & 0 \\ 0 & T \end{pmatrix},$$

and hence the TQ factorization of (8) is available trivially from T and $Q_{\rm FR}$ without additional expense.

The matrix Z_A is not kept fixed, since its role is purely to define an appropriate null space; the TQ factorization can therefore be updated in the normal fashion as the iterations proceed. No work is required to 'delete' the artificial constraints associated with Z_A when $Z_R^T g_{\rm FR} = 0$, since this simply involves repartitioning $Q_{\rm FR}$. The 'artificial' multiplier vector associated with the rows of Z_A^T is equal to $Z_A^T g_{\rm FR}$, and the multipliers corresponding to the rows of the 'true' working set are the multipliers that would be obtained if the artificial constraints were not present. If an artificial constraint is 'deleted' from the working set, an `A` appears alongside the entry in the `Jdel` column of the printed output (see Section 7 of the procedure document for `nag_qp_sol`).

The number of columns in Z_A and Z_R, the Euclidean norm of $Z_R^T g_{\rm FR}$, and the condition estimator of R appear in the printed output as `Art`, `Zr`, `Norm Gz` and `Cond Rz` respectively (see Section 7 of the procedure document for `nag_qp_sol`).

Under some circumstances, a different type of artificial constraint is used when solving a linear program. Although the algorithm of `nag_qp_sol` does not usually perform simplex steps (in the traditional sense), there is one exception: a linear program with fewer general constraints than variables (i.e., $n_{\rm L} \leq n$). (Use of the simplex method in this situation leads to savings in storage.) At the starting point, the 'natural' working set (the set of constraints exactly or nearly satisfied at the starting point) is augmented with a suitable number of 'temporary' bounds, each of which has the effect of temporarily fixing a variable at its current value. In subsequent iterations, a temporary bound is treated as a standard constraint until it is deleted from the working set, in which case it is never added again. If a temporary bound is 'deleted' from the working set, an `F` (for 'Fixed') appears alongside the entry in the `Jdel` column of the printed output (see Section 7 of the procedure document for `nag_qp_sol`).

References

[1] Gill P E, Hammarling S, Murray W, Saunders M A and Wright M H (1986) User's guide for LSSOL (Version 1.0) *Report SOL 86-1* Department of Operations Research, Stanford University

[2] Gill P E and Murray W (1978) Numerically stable methods for quadratic programming *Math. Programming* **14** 349–372

[3] Gill P E, Murray W, Saunders M A and Wright M H (1984) Procedures for optimization problems with a mixture of bounds and general linear constraints *ACM Trans. Math. Software* **10** 282–298

[4] Gill P E, Murray W, Saunders M A and Wright M H (1989) A practical anti-cycling procedure for linearly constrained optimization *Math. Programming* **45** 437–474

[5] Gill P E, Murray W, Saunders M A and Wright M H (1991) Inertia-controlling methods for general quadratic programming *SIAM Rev.* **33** 1–36

[6] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

[7] Pardalos P M and Schnitger G (1988) Checking local optimality in constrained quadratic programming is NP-hard *Operations Research Letters* **7** 33–35