Module 19.1: nag_ip
Integer Programming

`nag_ip` contains a procedure for solving 'zero-one', 'general', 'mixed' or 'all' integer linear programming problems.

Contents

Introduction

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

- `nag_ip_sol` computes a constrained minimum of a linear objective function subject to a set of general linear constraints and/or bounds on the variables when some (or all) of the variables are restricted to take integer values only. It may also be used to find a feasible integer point, the first integer solution or the optimal integer solution. It treats all matrices as dense and hence is not intended for large sparse problems.
- `nag_ip_cntrl_init` assigns default values to all the components of a structure of the derived type `nag_ip_cntrl_`*wp*.
- `nag_ip_cntrl_`*wp* may be used to supply optional parameters to `nag_ip_sol`.

Procedure: nag_ip_sol

1 Description

nag_ip_sol is designed to solve a certain type of integer programming (IP) problem — minimizing a linear function subject to constraints on the variables when some (or all) of the variables are restricted to take integer values only.

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

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

where 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.

The vector c may be zero, in which case the objective function is omitted and nag_ip_sol attempts to find a feasible point for the set of constraints.

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

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 $+\infty$.

If it is required that some (or all) of the variables in (1) are restricted to take integer values only, then the integer program is of type *mixed* (or *all*) general IP problem. If the integer variables are further restricted to take only 0−1 values (i.e., $l_j = 0$ and $u_j = 1$), then the integer program is of type (mixed or all) *zero-one* IP problem.

The branch and bound method (B&B) used by nag_ip_sol may be applied directly to such IP problems as follows. The general idea of B&B (see Dakin [1] or Mitra [2]) is to solve the problem without the integral restrictions as an LP problem (first *node*). If an integer variable x_k takes a non-integer value x_k^* in the optimal LP solution, two LP sub-problems are created by imposing $x_k \leq [x_k^*]$ and $x_k \geq [x_k^*] + 1$ respectively, where $[x_k^*]$ denotes the integer part of x_k^*. This process (known as *branching*) continues until the first integer solution (*bound*) is obtained. The hanging nodes are then solved and investigated in order to prove the optimality of the solution. At each node an LP problem is solved using lower-level procedures from nag_qp_sol.

Several options are available for controlling the operation of nag_ip_sol, 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_ip_cntrl_*wp*.

2 Usage

```
USE nag_ip
CALL nag_ip_sol(x, c, 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 original LP solution.

Output: the point at which `nag_ip_sol` terminated.

If **x**(j) is an integer variable for some j, $c \neq 0$ and

`error%code` = 0, **x** contains an estimate of either the optimal integer solution or first integer solution (depending on the value of the optional argument `first_int_sol`);

`error%code` = 101, **x** contains an estimate of the best integer solution found after searching the number of nodes specified by the optional argument `max_nodes`;

`error%code` = 102, **x** contains a point that is an integer solution but which is not optimal.

If **x**(j) is an integer variable for some j, $c = 0$ and `error%code` = 0, **x** contains a feasible integer point for the set of constraints.

If **x**(j) is a non-integer variable, for $j = 1, 2, \ldots, n$, `error%code` = 0 and

$c \neq 0$, **x** contains an estimate of the solution to the original LP problem;

$c = 0$, **x** contains a feasible point for the set of constraints.

c(n) — real(kind=*wp*), intent(in)

Input: the coefficients of the vector c of the objective function. To find a feasible integer point for the set of constraints, set **c** to zero. To find a feasible point for the set of constraints with no integer restrictions on the variables, set **c** to zero and the optional argument `non_int_var` to `.true.`.

obj_f — real(kind=*wp*), intent(out)

Output: the value of the objective function at the point returned in **x**.

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.

non_int_var(n) — logical, intent(in), optional

Input: specifies the non-integer and integer variables in the problem.

If `non_int_var`(j) = `.true.`, then **x**(j) is a non-integer variable;

if `non_int_var`(j) = `.false.` (the default), then **x**(j) is an integer variable.

Default: `non_int_var` = `.false.`.

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

x_upper(n) — real(kind=*wp*), intent(inout), 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_ip_cntrl_*wp*).

Constraints:

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

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

Output: if non_int_var(j) = .false. for some j and error%code = 0, 101 or 102, x_lower and x_upper contain the lower and upper bounds imposed on the IP solution or feasible integer point returned in x.

Default: x_lower = 0; 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_ip_cntrl_*wp*).

Constraints:

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

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

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

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

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

Output: if non_int_var(j) = .false. for some j and error%code = 0, 101 or 102, x_state contains the status of the bound constraints in the working set at the IP solution or feasible integer point returned in x. If non_int_var = .true., x_state contains the status of the bound constraints 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.

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

Output: if non_int_var(j) = .false. for some j and error%code = 0, 101 or 102, ax_state contains the status of the linear constraints in the working set at the IP solution or feasible integer point returned in x. If non_int_var = .true., ax_state contains the status of the linear constraints 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.
4	This corresponds to optimality being declared with x(j) being temporarily fixed at its current value.

max_depth — integer, intent(in), optional

Input: the maximum depth of the B&B tree to be searched.

Constraints: max_depth ≥ 2.

Default: max_depth = $\min(50, 5n)$.

max_nodes — integer, intent(in), optional

Input: the maximum number of nodes of the B&B tree to be searched. If max_nodes ≤ 0 and first_int_sol = .false., then the B&B tree search is continued until all nodes have been investigated.

Default: max_nodes = 0.

first_int_sol — logical, intent(in), optional

Input: specifies whether to terminate or continue the B&B tree search after the first integer solution (if any) has been found.

If first_int_sol = .true., then the B&B tree search is terminated at node k say, which contains the first integer solution. For max_nodes > 0, this applies only if $k \leq$ max_nodes.

If first_int_sol = .false., then the B&B tree search is continued after the first integer solution (if any) has been found.

Default: first_int_sol = .false..

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

Output: if
non_int_var(j) = .false. for some j and error%code = 0, 101 or 102, x_lambda contains the values of the Lagrange multipliers (*reduced costs*) for the bound constraints on the variables with respect to the current working set at the IP solution or feasible integer point returned in x. If non_int_var = .true., x_lambda contains the values of the multipliers for the bound constraints on the variables with respect to the current working set at the point returned in x.

More precisely, 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: if non_int_var(j) = .false. for some j and error%code = 0, 101 or 102, ax_lambda contains the values of the Lagrange multipliers (*shadow costs*) for the linear constraints with respect to the current working set at the IP solution or feasible integer point returned in x. If non_int_var = .true., ax_lambda contains the values of the multipliers for the linear constraints with respect to the current working set at the point returned in x.

More precisely, 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 and zero if ax_state(j) = 4.

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

control — type(nag_ip_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_ip_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.
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 to the original LP problem 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_ip_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})). Relax the integer restrictions in the problem and attempt to find the optimal LP solution by calling nag_qp_sol instead.
202	No feasible point was found for the original LP problem, i.e., it was not possible to satisfy all the constraints to within the feasibility tolerance.

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_ip_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}. You should also check that there are no constraint redundancies.

Alternatively, relax the integer restrictions in the problem and attempt to find the optimal LP solution by calling `nag_qp_sol` instead.

203 No feasible integer point was found, i.e., it was not possible to satisfy all the integer variables to within the integer feasibility tolerance.

The value of `control%int_feas_tol` (default value = `SQRT(EPSILON(1.0_wp))`; see the type definition for `nag_ip_cntrl_wp`) is too small. Rerun `nag_ip_sol` with a larger value.

204 No feasible integer point was found for the number of nodes investigated in the B&B tree.

The values of `control%int_feas_tol` (default value = `SQRT(EPSILON(1.0_wp))`; see the type definition for `nag_ip_cntrl_wp`) and/or the optional argument `max_nodes` may be too small. Either rerun `nag_ip_sol` with larger values or use the default values of `max_nodes` and the optional argument `first_int_sol` so that the B&B tree search is continued until all nodes have been investigated.

Warnings (error%level = 1):

error%code	Description

101 The IP solution reported is the best IP solution found for the number of nodes investigated in the B&B tree.

Rerun `nag_ip_sol` with a larger value of the optional argument `max_nodes` if you wish to attempt to improve upon the best IP solution found from a previous call to solve the same problem.

102 The IP solution reported is not optimal.

This occurs if the B&B tree search for at least one of the branches had to be terminated because an LP sub-problem in the branch did not have a solution.

The value of `control%iter_lim` (default value = max(50, $5\times$ $(n+n_{\mathrm{L}})$); see the type definition for `nag_ip_cntrl_wp`) may be too small. Rerun `nag_ip_sol` with a larger value.

103 The limiting number of iterations was reached before normal termination occurred for the original LP problem.

The value of `control%iter_lim` (default value = max(50, $5\times$ $(n+n_{\mathrm{L}})$); see the type definition for `nag_ip_cntrl_wp`) may be too small. Either rerun `nag_ip_sol` with a larger value or relax the integer restrictions in the problem and attempt to find the optimal LP solution by calling `nag_qp_sol` instead.

104 The maximum depth of the B&B tree is too small.

Rerun `nag_ip_sol` with a larger value of the optional argument `max_depth`.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 2 of this module document.

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.* [3] for further information and advice.

6.2 Accuracy

nag_ip_sol 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

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_ip_cntrl_*wp*) and rerunning the program. If the message recurs even after this change, you should relax the integer restrictions in the problem and attempt to find the optimal LP solution by calling nag_qp_sol instead.

7 Description of Printed Output

This section describes the intermediate and final printout produced by nag_ip_sol. 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 nag_ip_sol 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 = 1 or 10, the final printout (< 80 characters) at the end of execution of nag_ip_sol includes a listing of the status of every variable and constraint.

The following describes the printout for each variable.

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_ip_cntrl_*wp*), State will be ++ or -- respectively.
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_ip_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 or TF. If x is optimal, the multiplier should be non-negative if State is LL, and non-positive if State is UL.
Residual	is the difference between the variable Value and the nearer of its bounds x_lower(j) and x_upper(j).

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 if non_int_var(j) = .false. for some j, then the printed values of Lower Bound and Upper Bound for the jth variable may not be the same as those originally supplied in x_lower(j) and x_upper(j).

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

If non_int_var(j) = .false. for some j and control%print_level = 5 or 10, then the intermediate printout (< 80 characters) at the end of every node investigated during the execution of nag_ip_sol is a listing of the outcome of forcing an integer variable with a non-integer value to take a value within its specified upper and lower bounds.

The following describes the printout for each node investigated.

Node No	is the current node number of the B&B tree being investigated.
Parent Node	is the parent node number of the current node.
Obj Value	is the final objective value. This will be zero if a feasible integer point is being sought. If a node does not have a feasible solution, then No Feas Soln is printed instead of the objective function value. If a node whose optimal solution exceeds the best integer solution found so far is encountered (i.e., it does not pay to explore the sub-problem any further), then its objective function value is printed together with CO (Cut Off).
Varbl Chosen	is the index of the integer variable chosen for branching.
Value Before	is the non-integer value of the integer variable chosen for branching.
Lower Bound	is the lower bound value that the integer variable is allowed to take.
Upper Bound	is the upper bound value that the integer variable is allowed to take.
Value After	is the value of the integer variable after the current optimization.
Depth	is the depth of the B&B tree at the current node.

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

Procedure: nag_ip_cntrl_init

1 Description

nag_ip_cntrl_init assigns default values to the components of a structure of the derived type nag_ip_cntrl_*wp*.

2 Usage

```
USE nag_ip
CALL nag_ip_cntrl_init(control)
```

3 Arguments

3.1 Mandatory Argument

control — type(nag_ip_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_ip_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_ip_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_ip_cntrl_dp`. For single precision the name is `nag_ip_cntrl_sp`. Please read the Users' Note for your implementation to check which precisions are available.

1 Description

A structure of type `nag_ip_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_ip_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_ip_cntrl_wp
  !  Printing parameters
  logical ::  list
  integer ::  unit
  integer ::  print_level
  !  Algorithm choice and tolerances
  real(kind=wp) ::  feas_tol
  real(kind=wp) ::  inf_bound
  real(kind=wp) ::  inf_step
  real(kind=wp) ::  int_feas__tol
  integer ::  iter_lim
end type nag_ip_cntrl_wp
```

3 Components

3.1 Printing Parameters

list — logical

Controls the printing of the parameter settings in the call to `nag_ip_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_ip_sol` is sent.

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

Constraints: a valid output unit.

print_level — integer

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

If `non_int_var`(j) = `.false.` for some j and `c`$(k) \neq$ `0.0` for some k, then the following output is sent to the Fortran unit number defined by `unit`:

0	No output.
1	The final IP solution only.
5	One line of output (< 80 characters) for each node investigated and the final IP solution.
10	The original LP solution (first node), one line of output for each node investigated and the final IP solution.

If `non_int_var` = `.true.` and `c`$(k) \neq$ `0.0` for some k, then the following output is sent to the Fortran unit number defined by `unit`:

0	No output.
10	The original LP solution (first node) only.

If `non_int_var`(j) = `.false.` for some j and `c` = `0.0`, then the following output is sent to the Fortran unit number defined by `unit`:

0	No output.
1	The final feasible integer point only.
5	One line of output (< 80 characters) for each node investigated and the final feasible integer point.
10	The final feasible point for the original LP problem, one line of output for each node investigated and the final feasible integer point.

If `non_int_var` = `.true.` and `c` = `0.0`, then the following output is sent to the Fortran unit number defined by `unit`:

0	No output.
10	The final feasible point for the original LP problem.

Default: `print_level` = 10.

Constraints: `print_level` = 0, 1, 5 or 10.

3.2 Algorithm choice and tolerances

feas_tol — real(kind=*wp*)

`feas_tol` defines the maximum acceptable *absolute* violation in each constraint at a 'feasible' point. More precisely, a constraint is considered 'satisfied' if its violation does not exceed `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. 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`.

int_feas_tol — real(kind=*wp*)

`int_feas_tol` defines the maximum acceptable *absolute* violation in each variable at a 'feasible' integer point. For example, if the integer variable x_j is of order unity, then x_j is considered to be integer only if $(1.0 - \texttt{int\_feas\_tol}) \leq x_j \leq (1.0 + \texttt{int\_feas\_tol})$.

Default: `int_feas_tol` = `SQRT(EPSILON(1.0_wp))`.

Constraints: `int_feas_tol` $\geq$ `EPSILON(1.0_wp)`.

iter_lim — integer

`iter_lim` specifies the maximum number of iterations allowed before termination for each LP problem.

Default: `iter_lim` = max(50, 5×(no. of variables + no. of linear constraints)).

Constraints: `iter_lim` ≥ 1.

Example 1: All General Integer Programming Problem

To maximize the linear function

$$F(x) = 3x_1 + 4x_2$$

subject to the bounds

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

and to the linear constraints

$$\begin{aligned} 2x_1 + 5x_2 &\leq 15 \\ 2x_1 - 2x_2 &\leq 5 \\ 3x_1 + 2x_2 &\geq 5 \end{aligned}$$

where x_1 and x_2 are integer variables.

The initial point, which is feasible, is

$$x^{(0)} = (1,\ 1)^T.$$

The optimal solution is

$$x^* = (2,\ 2)^T,$$

and $F(x^*) = 14$.

Note that maximizing $F(x)$ is equivalent to minimizing $-F(x)$.

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_ip_ex01

  ! Example Program Text for nag_ip
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_ip, ONLY : nag_ip_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(:), x(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_ip_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 (x(n),c(n),a(m,n),ax_lower(m),ax_upper(m)) ! Allocate storage
```

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

      ! Solve the problem

      CALL nag_ip_sol(x,c,obj_f,a=a,ax_lower=ax_lower,ax_upper=ax_upper)

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

    END PROGRAM nag_ip_ex01
```

2 Program Data

```
Example Program Data for nag_ip_ex01
 3  2                                                    : m and n
  2.0       5.0
  2.0      -2.0
  3.0       2.0                                         : a
 -1.0e+25  -1.0e+25  5.0                                 : ax_lower
 15.0       5.0      1.0e+25                             : ax_upper
  1.0       1.0                                         : x
 -3.0      -4.0                                         : c
```

3 Program Results

```
 Example Program Results for nag_ip_ex01

 Parameters
 ----------

 Problem type...........        IP     (Integer Programming)

 linear constraints.....         3     variables..............         2
 integer variables......         2     non-integer variables..         0

 list...................    .true.     unit...................         6
 print_level............        10

 feas_tol...............  1.49E-08     int_feas_tol...........  1.49E-08
 inf_bound..............  1.00E+20     inf_step...............  1.00E+20
 iter_lim...............        50     eps (machine precision)  2.22E-16

 max_nodes..............         0     first_int_sol..........   .false.
 max_depth..............        10

   *** Optimal LP solution ***   -17.50000


 Varbl State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual

 V  1    FR    3.92857       0.00000         None        0.000       3.929
 V  2    FR    1.42857       0.00000         None        0.000       1.429


 L Con State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual
```

```
L  1    UL    15.0000         None        15.0000      -1.000       0.000
L  2    UL    5.00000         None        5.00000     -0.5000     -8.8818E-16
L  3    FR    14.6429       5.00000         None        0.000       9.643


  *** Start of tree search ***


 Node  Parent    Obj       Varbl  Value      Lower     Upper     Value    Depth
  No    Node    Value      Chosen Before     Bound     Bound     After
   2     1    No Feas Soln    1    3.93      4.00       None      4.00       1
   3     1     -16.2          1    3.93      0.00       3.00      3.00       1
   4     3     -15.5          2    1.80      2.00       None      2.00       2
   5     3     -13.0          2    1.80      0.00       1.00      1.00       2
     *** Integer solution ***


 Node  Parent    Obj       Varbl  Value      Lower     Upper     Value    Depth
  No    Node    Value      Chosen Before     Bound     Bound     After
   6     4    No Feas Soln    1    2.50      3.00       3.00      3.00       3
   7     4     -14.8          1    2.50      0.00       2.00      2.00       3
   8     7     -12.0     CO   2    2.20      3.00       None      3.00       4
   9     7     -14.0          2    2.20      2.00       2.00      2.00       4
     *** Integer solution ***

    *** End of tree search ***

Total of     9 nodes investigated.


Varbl State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual

V  1    UL    2.00000       0.00000       2.00000      -3.000       0.000
V  2    EQ    2.00000       2.00000       2.00000      -4.000       0.000


L Con State     Value     Lower Bound   Upper Bound    Lagr Mult   Residual

L  1    FR    14.0000         None        15.0000       0.000       1.000
L  2    FR    0.00000         None        5.00000       0.000       5.000
L  3    FR    10.0000       5.00000         None        0.000       5.000

Exit nag_ip_sol - Optimal IP solution found.

Final IP objective value =   -14.00000

Exit from nag_ip_sol after     6 iterations.
```

Example 2: Zero-one Integer Programming Problem

To maximize the linear function

$$F(x) = 3x_1 + 2x_2 - 5x_3 - 2x_4 + 3x_5$$

subject to the bounds

$$\begin{array}{rcl} 0 & \leq & x_1 \leq 1 \\ 0 & \leq & x_2 \leq 1 \\ 0 & \leq & x_3 \leq 1 \\ 0 & \leq & x_4 \leq 1 \\ 0 & \leq & x_5 \leq 1 \end{array}$$

and to the linear constraints

$$\begin{array}{rcrcrcrcl} x_1 & + & x_2 & + & x_3 & + & 2x_4 & + & x_5 \leq 4 \\ 7x_1 & & & + & 3x_3 & - & 4x_4 & + & 3x_5 \leq 8 \\ 11x_1 & - & 6x_2 & & & + & 3x_4 & - & 3x_5 \geq 3 \end{array}$$

where x_1, x_2, x_3, x_4 and x_5 are zero-one integer variables.

The initial point, which is infeasible, is

$$x^{(0)} = (1,\ 1,\ 1,\ 1,\ 1)^T.$$

The optimal solution is

$$x^* = (1,\ 1,\ 0,\ 0,\ 0)^T,$$

and $F(x^*) = 5$.

Note that maximizing $F(x)$ is equivalent to minimizing $-F(x)$.

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_ip_ex02

  ! Example Program Text for nag_ip
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_ip, ONLY : nag_ip_sol, nag_ip_cntrl_init, &
   nag_ip_cntrl_wp => nag_ip_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_ip_cntrl_wp) :: control
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), ax_lower(:), ax_upper(:), c(:), x(:), &
   x_upper(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_ip_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 (x(n),c(n),x_upper(n),a(m,n),ax_lower(m), &
       ax_upper(m))                ! Allocate storage

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

      ! Initialize control structure and set required control parameters

      CALL nag_ip_cntrl_init(control)

      control%print_level = 1
      control%iter_lim = 99

      ! Solve the problem

      CALL nag_ip_sol(x,c,obj_f,x_upper=x_upper,a=a,ax_lower=ax_lower, &
       ax_upper=ax_upper,control=control)

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

    END PROGRAM nag_ip_ex02
```

2 Program Data

```
Example Program Data for nag_ip_ex02
 3  5                                                        : m and n
  1.0       1.0      1.0      2.0       1.0
  7.0       0.0      3.0     -4.0       3.0
 11.0      -6.0      0.0      3.0      -3.0                  : a
-1.0e+25  -1.0e+25  3.0                                     : ax_lower
  4.0       8.0      1.0e+25                                : ax_upper
  1.0       1.0      1.0      1.0       1.0                 : x
  1.0       1.0      1.0      1.0       1.0                 : x_upper
 -3.0      -2.0      5.0      2.0      -3.0                 : c
```

3 Program Results

```
 Example Program Results for nag_ip_ex02

 Parameters
 ----------

 Problem type...........        IP     (Integer Programming)

 linear constraints.....         3     variables..............         5
 integer variables......         5     non-integer variables..         0

 list...................    .true.     unit...................         6
 print_level............         1
```

```
feas_tol.............  1.49E-08     int_feas_tol..........  1.49E-08
inf_bound............  1.00E+20     inf_step..............  1.00E+20
iter_lim.............        99     eps (machine precision)  2.22E-16

max_nodes............         0     first_int_sol.........   .false.
max_depth............        25
```

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
V 1	EQ	1.00000	1.00000	1.00000	-3.000	0.000
V 2	UL	1.00000	0.00000	1.00000	-2.000	0.000
V 3	LL	0.00000	0.00000	1.00000	5.000	0.000
V 4	EQ	0.00000	0.00000	0.00000	2.000	0.000
V 5	EQ	0.00000	0.00000	0.00000	-3.000	0.000

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
L 1	FR	2.00000	None	4.00000	0.000	2.000
L 2	FR	7.00000	None	8.00000	0.000	1.000
L 3	FR	5.00000	3.00000	None	0.000	2.000

```
Exit nag_ip_sol - Optimal IP solution found.

Final IP objective value =   -5.000000

Exit from nag_ip_sol after    15 iterations.
```

References

[1] Dakin R J (1965) A tree search algorithm for mixed integer programming problems *Comput. J.* **8** 250–255

[2] Mitra G (1973) Investigation of some branch and bound strategies for the solution of mixed integer linear programs *Math. Programming* **4** 155–170

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