NAG Toolbox

nag_mip_sqp (h02da)

1 Purpose

nag_mip_sqp (h02da) solves general nonlinear programming problems with integer constraints on some of the variables.

2 Syntax

```
[ax, x, c, cjac, objgrd, objmip, user, ifail] = nag_mip_sqp(ncnln, a, d, bl, bu, varcon, x, confun, objfun, iopts, opts, 'n', n, 'nclin', nclin, 'maxit', maxit, 'acc', acc, 'user', user)
[ax, x, c, cjac, objgrd, objmip, user, ifail] = h02da(ncnln, a, d, bl, bu, varcon, x, confun, objfun, iopts, opts, 'n', n, 'nclin', nclin, 'maxit', maxit, 'acc', acc, 'user', user)
```

Before calling nag_mip_sqp (h02da), nag_mip_optset (h02zk) **must** be called with **optstr** set to ‘’. Optional parameters may also be specified by calling nag_mip_optset (h02zk) before the call to nag_mip_sqp (h02da).

3 Description

nag_mip_sqp (h02da) solves mixed integer nonlinear programming problems using a modified sequential quadratic programming method. The problem is assumed to be stated in the following general form:

$$\begin{array}{ll} \underset{x\in\{R^{n_c},Z^{n_i}\}}{\text{minimize}} & f(x) \\ \text{subject to} & c_j(x) = 0, \quad j = 1,2,\ldots,m_e \\ & c_j(x) \geq 0, \quad j = m_e+1, m_e+2,\ldots,m \\ & l \leq x_i \leq u, \quad i = 1,2,\ldots,n \end{array}$$

with n_c continuous variables and n_i binary and integer variables in a total of n variables; m_e equality constraints in a total of m constraint functions.

Partial derivatives of $f(x)$ and $c(x)$ are not required for the n_i integer variables. Gradients with respect to integer variables are approximated by difference formulae.

No assumptions are made regarding $f(x)$ except that it is twice continuously differentiable with respect to continuous elements of x. It is not assumed that integer variables are relaxable. In other words, problem functions are evaluated only at integer points.

The method seeks to minimize the exact penalty function:

$$P_\sigma(x) = f(x) + \sigma\|g(x)\|_\infty$$

where σ is adapted by the algorithm and $g(x)$ is given by:

$$\begin{array}{rcll} g(x) & = & c_j(x), & j = 1,2,\ldots,m_e \\ & = & \min\big(c_j(x),0\big), & j = m_e+1, m_e+2,\ldots,m. \end{array}$$

Successive quadratic approximations are applied under the assumption that integer variables have a smooth influence on the model functions, that is function values do not change drastically when incrementing or decrementing an integer value. In practice this requires integer variables to be ordinal not categorical. The algorithm is stabilised by a trust region method including Yuan's second order corrections, see Yuan and Sun (2006). The Hessian of the Lagrangian function is approximated by BFGS (see Section 11.4 in nag_opt_nlp1_solve (e04uc)) updates subject to the continuous and integer variables.

The mixed-integer quadratic programming subproblems of the SQP-trust region method are solved by a branch and cut method with continuous subproblem solutions obtained by the primal-dual method of Goldfarb and Idnani, see Powell (1983). Different strategies are available for selecting a branching variable:

Maximal fractional branching. Select an integer variable from the relaxed subproblem solution with largest distance from next integer value

Minimal fractional branching. Select an integer variable from the relaxed subproblem solution with smallest distance from next integer value

and a node from where branching, that is the generation of two new subproblems, begins:

Best of two. The optimal objective function values of the two child nodes are compared and the node with a lower value is chosen

Best of all. Select an integer variable from the relaxed subproblem solution with the smallest distance from the next integer value

Depth first. Select a child node whenever possible.

This implementation is based on the algorithm MISQP as described in Exler *et al.* (2013).

Linear constraints may optionally be supplied by a matrix A and vector d rather than the constraint functions $c(x)$ such that

$$Ax = d \quad \text{or} \quad Ax \geq d.$$

Partial derivatives with respect to x of these constraint functions are not requested by nag_mip_sqp (h02da).

4 References

Exler O, Lehmann T and Schittkowski K (2013) A comparative study of SQP-type algorithms for nonlinear and nonconvex mixed-integer optimization *Mathematical Programming Computation* **4** 383–412

Mann A (1986) GAMS/MINOS: Three examples *Department of Operations Research Technical Report* Stanford University

Powell M J D (1983) On the quadratic programming algorithm of Goldfarb and Idnani *Report DAMTP 1983/Na 19* University of Cambridge, Cambridge

Yuan Y-x and Sun W (2006) *Optimization Theory and Methods* Springer–Verlag

5 Parameters

5.1 Compulsory Input Parameters

1: **ncnln** – INTEGER

n_N, the number of constraints supplied by $c(x)$.

Constraint: **ncnln** ≥ 0.

2: **a**$(lda, :)$ – REAL (KIND=nag_wp) array

The first dimension of the array **a** must be at least $\max(1, \mathbf{nclin})$.

The second dimension of the array **a** must be at least **n** if **nclin** > 0.

The ith row of **a** must contain the coefficients of the ith general linear constraint, for $i = 1, 2, \ldots, n_l$. Any equality constraints must be specified first.

If **nclin** $= 0$, the array **a** is not referenced.

3: **d**(**nclin**) – REAL (KIND=nag_wp) array

d_i, the constant for the ith linear constraint.

If **nclin** $= 0$, the array **d** is not referenced.

4: **bl**(**n**) – REAL (KIND=nag_wp) array

5: **bu**(**n**) – REAL (KIND=nag_wp) array

bl must contain the lower bounds, l_i, and **bu** the upper bounds, u_i, for the variables; bounds on integer variables are rounded, bounds on binary variables need not be supplied.

Constraint: **bl**$(i) \leq$ **bu**(i), for $i = 1, 2, \ldots,$ **n**.

6: **varcon**(**n** + **nclin** + **ncnln**) – INTEGER array

varcon indicates the nature of each variable and constraint in the problem. The first n elements of the array must describe the nature of the variables, the next n_L elements the nature of the general linear constraints (if any) and the next n_N elements the general constraints (if any).

varcon$(j) = 0$
A continuous variable.

varcon$(j) = 1$
A binary variable.

varcon$(j) = 2$
An integer variable.

varcon$(j) = 3$
An equality constraint.

varcon$(j) = 4$
An inequality constraint.

Constraints:

varcon$(j) = 0$, 1 or 2, for $j = 1, 2, \ldots,$ **n**;
varcon$(j) = 3$ or 4, for $j =$ **n** $+ 1, \ldots,$ **n** + **nclin** + **ncnln**;
At least one variable must be either binary or integer;
Any equality constraints must precede any inequality constraints.

7: **x**(**n**) – REAL (KIND=nag_wp) array

An initial estimate of the solution, which need not be feasible. Values corresponding to integer variables are rounded; if an initial value less than half is supplied for a binary variable the value zero is used, otherwise the value one is used.

8: **confun** – SUBROUTINE, supplied by the NAG Library or the user.

confun must calculate the constraint functions supplied by $c(x)$ and their Jacobian at x. If all constraints are supplied by A and d (i.e., **ncnln** $= 0$), **confun** will never be called by nag_mip_sqp (h02da) and **confun** may be the string `nag_mip_sqp_dummy_confun (h02ddm)`. (nag_mip_sqp_dummy_confun (h02ddm) is included in the NAG Toolbox.) If **ncnln** > 0, the first call to **confun** will occur after the first call to **objfun**.

```
[mode, c, cjac, user] = confun(mode, ncnln, n, varcon, x, cjac, nstate,
user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$
Elements of **c** containing continuous variables.

mode $= 1$
Elements of **cjac** containing continuous variables.

2: **ncnln** – INTEGER

The dimension of the array **c** and the first dimension of the array **cjac**. the number of constraints supplied by $c(x)$, n_N.

3: **n** – INTEGER

The second dimension of the array **cjac**. n, the total number of variables, $n_c + n_i$.

4: **varcon**$(\mathbf{n} + nclin + \mathbf{ncnln})$ – INTEGER array

The array **varcon** as supplied to nag_mip_sqp (h02da).

5: **x**$(\mathbf{n})$ – REAL (KIND=nag_wp) array

The vector of variables at which the objective function and/or all continuous elements of its gradient are to be evaluated.

6: **cjac**$(\mathbf{ncnln}, \mathbf{n})$ – REAL (KIND=nag_wp) array

Note: the derivative of the ith constraint with respect to the jth variable, $\dfrac{\partial c_i}{\partial x_{\mathrm{j}}}$, is stored in **cjac**$(i, j)$.

Continuous elements of **cjac** are set to the value of NaN.

7: **nstate** – INTEGER

If **nstate** $= 1$, nag_mip_sqp (h02da) is calling **confun** for the first time. This argument setting allows you to save computation time if certain data must be read or calculated only once.

8: **user** – INTEGER array

confun is called from nag_mip_sqp (h02da) with the object supplied to nag_mip_sqp (h02da).

Output Parameters

1: **mode** – INTEGER

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case nag_mip_sqp (h02da) will terminate with **ifail** set to **mode**.

2: **c**$(\mathbf{max}(\mathbf{1}, \mathbf{ncnln}))$ – REAL (KIND=nag_wp) array

Must contain **ncnln** constraint values, with the value of the jth constraint $c_j(x)$ in **c**(j).

3: **cjac**(**ncnln**, **n**) – REAL (KIND=nag_wp) array

Note: the derivative of the ith constraint with respect to the jth variable, $\dfrac{\partial c_i}{\partial x_{\mathrm{j}}}$, is stored in **cjac**$(i, j)$.

The ith row of **cjac** must contain elements of $\dfrac{\partial c_i}{\partial x_{\mathrm{j}}}$ for each continuous variable x_j.

4: **user** – INTEGER array

9: **objfun** – SUBROUTINE, supplied by the user.

objfun must calculate the objective function $f(x)$ and its gradient for a specified n-element vector x.

```
[mode, objmip, objgrd, user] = objfun(mode, n, varcon, x, objgrd, nstate,
user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$
The objective function value, **objmip**.

mode $= 1$
The continuous elements of **objgrd**.

2: **n** – INTEGER

n, the total number of variables, $n_c + n_i$.

3: **varcon**(**n** $+ nclin + ncnln$) – INTEGER array

The array **varcon** as supplied to nag_mip_sqp (h02da).

4: **x**(**n**) – REAL (KIND=nag_wp) array

The vector of variables at which the objective function and/or all continuous elements of its gradient are to be evaluated.

5: **objgrd**(**n**) – REAL (KIND=nag_wp) array

Continuous elements of **objgrd** are set to the value of NaN.

6: **nstate** – INTEGER

If **nstate** $= 1$, nag_mip_sqp (h02da) is calling **objfun** for the first time. This argument setting allows you to save computation time if certain data must be read or calculated only once.

7: **user** – INTEGER array

objfun is called from nag_mip_sqp (h02da) with the object supplied to nag_mip_sqp (h02da).

Output Parameters

1: **mode** – INTEGER

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case nag_mip_sqp (h02da) will terminate with **ifail** set to **mode**.

2: **objmip** – REAL (KIND=nag_wp)

Must be set to the objective function value, f, if **mode** $= 0$; otherwise **objmip** is not referenced.

3: **objgrd**(**n**) – REAL (KIND=nag_wp) array

Must contain the gradient vector of the objective function if **mode** $= 1$, with **objgrd**(j) containing the partial derivative of f with respect to continuous variable x_j; otherwise **objgrd** is not referenced.

4: **user** – INTEGER array

10: **iopts**(:) – INTEGER array

Note: the dimension of this array is dictated by the requirements of associated functions that must have been previously called. This array **must** be the same array passed as argument **iopts** in the previous call to nag_mip_optset (h02zk).

11: **opts**(:) – REAL (KIND=nag_wp) array

Note: the dimension of this array is dictated by the requirements of associated functions that must have been previously called. This array **must** be the same array passed as argument **opts** in the previous call to nag_mip_optset (h02zk).

The real option array as returned by nag_mip_optset (h02zk).

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **bl**, **bu**, **x** and the second dimension of the array **a**. (An error is raised if these dimensions are not equal.)

n, the total number of variables, $n_c + n_i$.

Constraint: **n** > 0.

2: **nclin** – INTEGER

Default: the dimension of the array **d** and the first dimension of the array **a**. (An error is raised if these dimensions are not equal.)

n_l, the number of general linear constraints defined by A and d.

Constraint: **nclin** ≥ 0.

3: **maxit** – INTEGER

Suggested value: **maxit** $= 500$.

Default: 500

The maximum number of iterations within which to find a solution. If **maxit** is less than or equal to zero, the suggested value below is used.

4: **acc** – REAL (KIND=nag_wp)

Suggested value: **acc** $= 0.0001$.

Default: 1.0e − 4

The requested accuracy for determining feasible points during iterations and for halting the method when the predicted improvement in objective function is less than **acc**. If **acc** is less than or equal to ϵ (ϵ being the ***machine precision*** as given by nag_machine_precision (x02aj)), the below suggested value is used.

5: **user** – INTEGER array

user is not used by nag_mip_sqp (h02da), but is passed to **confun** and **objfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

1: **ax**(**max**(**1**, **nclin**)) – REAL (KIND=nag_wp) array

The final values of the linear constraints Ax.

If **nclin** $= 0$, **ax** is not referenced.

2: **x**(**n**) – REAL (KIND=nag_wp) array

The final estimate of the solution.

3: **c**(**max**(**1**, **ncnln**)) – REAL (KIND=nag_wp) array

If **ncnln** > 0, **c**(j) contains the value of the jth constraint function $c_j(x)$ at the final iterate, for $j = 1, 2, \ldots,$ **ncnln**.

If **ncnln** $= 0$, the array **c** is not referenced.

4: **cjac**(**max**(**1**, **ncnln**), **n**) – REAL (KIND=nag_wp) array

Note: the derivative of the ith constraint with respect to the jth variable, $\dfrac{\partial c_i}{\partial x_{\mathrm{j}}}$, is stored in **cjac**$(i, j)$.

If **ncnln** > 0, **cjac** contains the Jacobian matrix of the constraint functions at the final iterate, i.e., **cjac**(i, j) contains the partial derivative of the ith constraint function with respect to the jth variable, for $i = 1, 2, \ldots,$ **ncnln** and $j = 1, 2, \ldots,$ **n**. (See the discussion of argument **cjac** under **confun**.)

If **ncnln** $= 0$, the array **cjac** is not referenced.

5: **objgrd**(**n**) – REAL (KIND=nag_wp) array

The objective function gradient at the solution.

6: **objmip** – REAL (KIND=nag_wp)

With **ifail** $= 0$, **objmip** contains the value of the objective function for the MINLP solution.

7: **user** – INTEGER array

8: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

Constraint: $\mathbf{n} > 0$.

ifail $= 2$

Constraint: **nclin** ≥ 0.

ifail $= 3$

Constraint: **ncnln** ≥ 0.

ifail $= 4$

Constraint: $lda \geq$ **nclin**.

ifail $= 5$

Constraint: $\mathbf{bl}(i) \leq \mathbf{bu}(i)$, for $i = 1, 2, \ldots, \mathbf{n}$.

ifail $= 6$

Constraint: $\mathbf{varcon}(i) = 0,\ 1$ or 2, for $i = 1, 2, \ldots, \mathbf{n}$.

ifail $= 7$

Constraint: $\mathbf{varcon}(i) = 3$ or 4, for $i = \mathbf{n} + 1, \ldots, \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$.

ifail $= 8$

The supplied **objfun** returned a NaN value.

ifail $= 9$

The supplied **confun** returned a NaN value.

ifail $= 10$

On entry, the optional parameter arrays **iopts** and **opts** have either not been initialized or been corrupted.

ifail $= 11$

On entry, there are no binary or integer variables.

ifail $= 12$

On entry, linear equality constraints do not precede linear inequality constraints.

ifail $= 13$

On entry, nonlinear equality constraints do not precede nonlinear inequality constraints.

ifail $= 81$

One or more objective gradients appear to be incorrect.

ifail $= 91$

One or more constraint gradients appear to be incorrect.

ifail $= 1001$

On entry. Exceeded the maximum number of iterations.

ifail $= 1002$

More than the maximum number of feasible steps without improvement in the objective function.

ifail $= 1003$

Penalty parameter tends to infinity in an underlying mixed-integer quadratic program; the problem may be infeasible.

ifail $= 1004$

Termination at an infeasible iterate; if the problem is feasible, try a different starting value.

ifail $= 1005$

Termination with zero integer trust region for integer variables; try a different starting value.

ifail $= 1008$

The optimization failed due to numerical difficulties. Set optional parameter **Print Level** $= 3$ for more information.

ifail < 0

The optimization halted because you set **mode** negative in **objfun** or **mode** negative in **confun**, to $\langle value \rangle$.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

Select a portfolio of at most p assets from n available with expected return ρ, is fully invested and that minimizes

$$
\begin{array}{lllll}
 & x^{\mathrm{T}}\Sigma x & & & \\
\text{subject to} & r^{\mathrm{T}}x & = & \rho & \\
 & \sum_{i=1}^{n} x_i & = & 1 & \\
 & x_i & \leq & y_i & \\
 & \sum_{i=1}^{n} y_i & \leq & p & \\
 & x_i & \geq & 0 & \\
 & y_i & = & 0 \quad \text{or} & \quad 1
\end{array}
$$

where

x is a vector of proportions of selected assets

y is an indicator variable that describes if an asset is in or out

r is a vector of mean returns

Σ is the covariance matrix of returns.

This example is taken from Mann (1986) with

$$
\begin{array}{lcl}
r & = & \begin{pmatrix} 8 & 9 & 12 & 7 \end{pmatrix} \\
\Sigma & = & \begin{pmatrix} 4 & 3 & -1 & 0 \\ 3 & 6 & 1 & 0 \\ -1 & 1 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\
p & = & 3 \\
\rho & = & 10.
\end{array}
$$

Linear constraints are supplied through both A and d, and **confun**.

9.1 Program Text

```
function h02da_example

diary h02da_example.r
fprintf('h02da example results\n\n');

% Problem size
n = nag_int(8);
nclin = nag_int(5);
ncnln = nag_int(2);

varcon = zeros(n+nclin+ncnln, 1, nag_int_name);

% Set variable types: continuous then binary
varcon(1:4) = nag_int([0;0;0;0]);
varcon(5:8) = nag_int([1;1;1;1]);

bl = zeros(n,1);
bu = zeros(n,1);

% Set bounds of continuous variables
bl(1:4) = 0;
bu(1:4) = 1000;

% Set linear constraints
varcon(n+1)         = 3; % equality
varcon(n+2:n+nclin) = 4; % inequality

% Set Ax=d then Ax>=d
```

```
  a = zeros(nclin,n);
  a(1,1:4) = 1;
  a(2,[1,5]) = [-1,1];
  a(3,[2,6]) = [-1,1];
  a(4,[3,7]) = [-1,1];
  a(5,[4,8]) = [-1,1];

  d = zeros(nclin,1);
  d(1) = 1;

  % Set constraints supplied by confun, equality first
  varcon(n+nclin+1) = 3;
  varcon(n+nclin+2) = 4;

  iopts = zeros(200, 1, nag_int_name);
  opts = zeros(100, 1);

  % Initialize options
  [iopts, opts, ifail] = h02zk( ...
                                'Initialize = h02da', iopts, opts);

  % Optimisation parameters
  maxit =  nag_int(500);
  acc = 1e-6;

  x = zeros(n,1);
  % Initial estimate (binary variables need not be given)
  x(1:4) = 1;

  % Portfolio parameters
  user.p = 3;
  user.rho = 10;

  % Call the solver
  [ax, x, c, cjac, objgrd, objmip, user, ifail] = ...
  h02da( ...
         ncnln, a, d, bl, bu, varcon, x, @confun, @objfun, iopts, opts, ...
         'user', user);

  % Query the accuracy of the mixed integer QP solver
  [ivalue, accqp, cvalue, optype, ifail] = ...
    h02zl('QP Accuracy', iopts, opts);

  % Results
  [ifail] = x04ca('G', ' ', x, 'Final estimate');
  fprintf('\nMixed integer QP Accuracy: %g', accqp);
  fprintf('\nOptimised value: %g\n\n', objmip);

  diary off

function [mode, c, cjac, user] = ...
         confun(mode, ncnln, n, varcon, x, cjac, nstate, user)
  c = zeros(ncnln, 1);

  if (mode == 0)
    % Constraints
    % Mean return at least rho:
    c(1) = 8*x(1) + 9*x(2) + 12*x(3) + 7*x(4) - user.rho;

    % Maximum of p assets in portfolio:
    c(2) = user.p - x(5) - x(6) - x(7) - x(8);
  else
    % Jacobian
    cjac(1,1:4) = [8,9,12,7];

    % c(2) does not include continuous variables which requires
    % that their derivatives are zero
    cjac(2,1:4) = 0;
  end

function [mode, objmip, objgrd, user] = ...
```

```
        objfun(mode, n, varcon, x, objgrd, nstate, user)
  objmip = 0;

  if (mode == 0)
    % Objective value
    objmip = x(1)*(4*x(1)+3*x(2)-x(3)) + ...
             x(2)*(3*x(1)+6*x(2)+x(3)) + ...
             x(3)*(x(2)-x(1)+10*x(3));
  else
    % Objective gradients for continuous variables
    objgrd(1) = 8*x(1) +  6*x(2) -  2*x(3);
    objgrd(2) = 6*x(1) + 12*x(2) +  2*x(3);
    objgrd(3) = 2*(x(2)-x(1)) + 20*x(3);
    objgrd(4) = 0;
  end
```

9.2 Program Results

```
     h02da example results

 Final estimate
          1
 1   0.3750
 2   0.0000
 3   0.5250
 4   0.1000
 5   1.0000
 6   0.0000
 7   1.0000
 8   1.0000

Mixed integer QP Accuracy: 1e-10
Optimised value: 2.925
```

10 Optional Parameters

This section can be skipped if you wish to use the default values for all optional parameters, otherwise, the following is a list of the optional parameters available and a full description of each optional parameter is provided in Section 11.1.

Branch Bound Steps

Branching Rule

Check Gradients

Continuous Trust Radius

Descent

Descent Factor

Feasible Steps

Improved Bounds

Integer Trust Radius

Maximum Restarts

Minor Print Level

Modify Hessian

Node Selection

Non Monotone

Objective Scale Bound

Penalty

Penalty Factor

Print Level

QP Accuracy

Scale Continuous Variables
Scale Objective Function
Warm Starts

10.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively.

All options accept the value DEFAULT in order to return single options to their default states.

Keywords and character values are case insensitive, however they must be separated by at least one space.

nag_mip_optset (h02zk) can be called to supply options, one call being necessary for each optional parameter. For example,

```
Call H02ZKF('Check Gradients = Yes', iopts, liopts, opts, lopts, ifail)
```

nag_mip_optset (h02zk) should be consulted for a full description of the method of supplying optional parameters.

For nag_mip_sqp (h02da) the maximum length of the argument **cvalue** used by nag_mip_optget (h02zl) is 12.

Branch Bound Steps i Default $= 500$

Maximum number of branch-and-bound steps for solving the mixed integer quadratic problems.

Constraint: **Branch Bound Steps** > 1.

Branching Rule a Default $=$ Maximum

Branching rule for branch and bound search.

Branching Rule $=$ Maximum
Maximum fractional branching.

Branching Rule $=$ Minimum
Minimum fractional branching.

Check Gradients a Default $=$ No

Perform an internal check of supplied objective and constraint gradients. It is advisable to set **Check Gradients** $=$ Yes during code development to avoid difficulties associated with incorrect user-supplied gradients.

Continuous Trust Radius r Default $= 10.0$

Initial continuous trust region radius, Δ_0^c; the initial trial step $d \in R^{n_c}$ for the SQP approximation must satisfy $\|d\|_\infty \leq \Delta_0^c$.

Constraint: **Continuous Trust Radius** > 0.0.

Descent r Default $= 0.05$

Initial descent parameter, δ, larger values of δ allow penalty parameter σ to increase faster which can lead to faster convergence.

Constraint: $0.0 <$ **Descent** < 1.0.

Descent Factor r Default $= 0.1$

Factor for decreasing the internal descent parameter, δ, between iterations.

Constraint: $0.0 <$ **Descent Factor** < 1.0.

Feasible Steps i Default $= 10$

Maximum number of feasible steps without improvements, where feasibility is measured by $\|g(x)\|_\infty \leq \sqrt{\textbf{acc}}$.

Constraint: **Feasible Steps** > 1.

Improved Bounds a Default $=$ No

Calculate improved bounds in case of 'Best of all' node selection strategy.

Integer Trust Radius r Default $= 10.0$

Initial integer trust region radius, Δ_0^i; the initial trial step $e \in R^{n_i}$ for the SQP approximation must satisfy $\|e\|_\infty \leq \Delta_0^i$.

Constraint: **Integer Trust Radius** ≥ 1.0.

Maximum Restarts i Default $= 2$

Maximum number of restarts that allow the mixed integer SQP algorithm to return to a better solution. Setting a value higher than the default might lead to better results at the expense of function evaluations.

Constraint: $0 <$ **Maximum Restarts** ≤ 15.

Minor Print Level i Default $= 0$

Print level of the subproblem solver. Active only if **Print Level** $\neq 0$.

Constraint: $0 <$ **Minor Print Level** < 4.

Modify Hessian a Default $=$ Yes

Modify the Hessian approximation in an attempt to get more accurate search directions. Calculation time is increased when the number of integer variables is large.

Node Selection a Default $=$ Depth First

Node selection strategy for branch and bound.

Node Selection $=$ Best of all
: Large tree search; this method is the slowest as it solves all subproblem QPs independently.

Node Selection $=$ Best of two
: Uses warm starts and less memory.

Node Selection $=$ Depth first
: Uses more warm starts. If warm starts are applied, they can speed up the solution of mixed integer subproblems significantly when solving almost identical QPs.

Non Monotone i Default $= 10$

Maximum number of successive iterations considered for the non-monotone trust region algorithm, allowing the penalty function to increase.

Constraint: $0 <$ **Non Monotone** < 100.

Objective Scale Bound r Default $= 1.0$

When **Scale Objective Function** > 0 internally scale absolute function values greater than 1.0 or **Objective Scale Bound**.

Constraint: **Objective Scale Bound** > 0.0.

Penalty r Default $= 1000.0$

Initial penalty parameter, σ.

Constraint: **Penalty** ≥ 0.0.

Penalty Factor r Default $= 10.0$

Factor for increasing penalty parameter σ when the trust regions cannot be enlarged at a trial step.

Constraint: **Penalty Factor** > 1.0.

Print Level i Default $= 0$

Specifies the desired output level of printing.

Print Level $= 0$
No output.

Print Level $= 1$
Final convergence analysis.

Print Level $= 2$
One line of intermediate results per iteration.

Print Level $= 3$
Detailed information printed per iteration.

QP Accuracy r Default $= 1.0\text{e}{-10}$

Termination tolerance of the relaxed quadratic program subproblems.

Constraint: **QP Accuracy** > 0.0.

Scale Continuous Variables a Default $=$ Yes

Internally scale continuous variables values.

Scale Objective Function i Default $= 1$

Internally scale objective function values.

Scale Objective Function $= 0$
No scaling.

Scale Objective Function $= 1$
Scale absolute values greater than **Objective Scale Bound**.

Warm Starts i Default $= 100$

Maximum number of warm starts within the mixed integer QP solver, see **Node Selection**.

Constraint: **Warm Starts** ≥ 0.