NAG Library Function Document

nag_opt_handle_solve_pennon (e04svc)

1 Purpose

nag_opt_handle_solve_pennon (e04svc) is a solver from the NAG optimization modelling suite for problems such as, quadratic programming (QP), linear semidefinite programming (SDP) and semidefinite programming with bilinear matrix inequalities (BMI-SDP).

2 Specification

```
#include <nag.h>
#include <nage04.h>
```

```
void nag_opt_handle_solve_pennon (void *handle, Integer nvar, double x[],
     Integer nnzu, double u[], Integer nnzuc, double uc[], Integer nnzua,
     double ua[], double rinfo[], double stats[], Integer *inform,
     NagError *fail)
```

3 Description

nag_opt_handle_solve_pennon (e04svc) serves as a solver for compatible problems stored as a handle. The handle points to an internal data structure which defines the problem and serves as means of communication for functions in the suite. First, the problem handle is initialized by nag_opt_handle_init (e04rac). Then some of the functions nag_opt_handle_set_linobj (e04rec), nag_opt_handle_set_quadobj (e04rfc), nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc), na g_opt_handle_set_linmatineq (e04rnc) or nag_opt_handle_set_quadmatineq (e04rpc) may be used to formulate the objective function, (standard) constraints and matrix constraints of the problem. Once the problem is fully set, the handle may be passed to the solver. When the handle is not needed anymore, nag_opt_handle_free (e04rzc) should be called to destroy it and deallocate the memory held within. See nag_opt_handle_init (e04rac) for more details.

Problems which can be defined this way are, for example, (generally nonconvex) quadratic programming (QP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & l_B \le Bx \le u_B & \text{(b)} \\ & l_x \le x \le u_x, & \text{(c)} \end{array} \qquad (1)$$

linear semidefinite programming problems (SDP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\dots,m_A & \text{(b)} \\ & l_B \le Bx \le u_B & \text{(c)} \\ & l_x \le x \le u_x & \text{(d)} \end{array} \qquad (2)$$

or semidefinite programming problems with bilinear matrix inequalities (BMI-SDP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\dots,m_A & \text{(b)} \\ & l_B \le Bx \le u_B & \text{(c)} \\ & l_x \le x \le u_x. & \text{(d)} \end{array} \qquad (3)$$

Here c, l_x and u_x are n-dimensional vectors, H is a symmetric n by n matrix, l_B, u_B are

m_B-dimensional vectors, B is a general m_B by n rectangular matrix and A_i^k, Q_{ij}^k are symmetric matrices. The expression $S \succeq 0$ stands for a constraint on eigenvalues of a symmetric matrix S, namely, all the eigenvalues should be non-negative, i.e., the matrix should be positive semidefinite. See relevant functions of the suite for more details on the problem formulation.

The solver is based on a generalized Augmented Lagrangian method with a suitable choice of standard and matrix penalty functions. For a detailed description of the algorithm see Section 11. Under standard assumptions on the problem (Slater constraint qualification, boundedness of the objective function on the feasible set, see Stingl (2006) for details) the algorithm converges to a local solution. In case of convex problems such as linear SDP or convex QP, this is the global solution. The solver is suitable for both small dense and large-scale sparse problems.

The algorithm behaviour and solver strategy can be modified by various optional parameters (see Section 12) which can be set by nag_opt_handle_opt_set (e04zmc) and nag_opt_handle_opt_set_file (e04zpc) anytime between the initialization of the handle by nag_opt_handle_init (e04rac) and a call to the solver. Once the solver has finished, options may be modified for the next solve. The solver may be called repeatedly with various starting points and/or optional parameters.

There are several optional parameters with a multiple choice where the default choice is AUTO (for example, **Hessian Density**). This value means that the decision over the option is left to the solver based on the structure of the problem. Option getter nag_opt_handle_opt_get (e04znc) can be called to retrieve the choice of these options as well as on any other options.

Optional parameter **Task** may be used to switch the problem to maximization or to ignore the objective function and find only a feasible point.

Optional parameter **Monitor Frequency** may be used to turn on the monitor mode of the solver. The solver invoked in this mode pauses regularly even before the optimal point is found to allow monitoring the progress from the calling program. All the important error measures and statistics are available in the calling program which may terminate the solver early if desired (see argument **inform**).

3.1 Structure of the Lagrangian Multipliers

The algorithm works internally with estimates of both the decision variables, denoted by x, and the Lagrangian multipliers (dual variables) for standard and matrix constraints, denoted by u and U, respectively. You may provide initial estimates, request approximations during the run (the monitor mode turned on) and obtain the final values. The Lagrangian multipliers are split into two arrays, the multipliers u for (standard) constraints are stored in array **u** and multipliers U for matrix constraints in array **ua**. Both arrays need to conform to the structure of the constraints.

If the simple bounds were defined (nag_opt_handle_set_simplebounds (e04rhc) was successfully called), the first $2n$ elements of **u** belong to the corresponding Lagrangian multipliers, interleaving a multiplier for the lower and for the upper bound for each x_i. If any of the bounds were set to infinity, the corresponding Lagrangian multipliers are set to 0 and may be ignored.

Similarly, the following $2m_B$ elements of **u** belong to multipliers for the linear constraints, if formulated by nag_opt_handle_set_linconstr (e04rjc). The organization is the same, i.e., the multipliers for each constraint for the lower and upper bounds are alternated and zeroes are used for any missing (infinite bound) constraint.

A Lagrangian multiplier for a matrix constraint (one block) of dimension d by d is a dense symmetric matrix of the same dimension. All multipliers U are stored next to each other in array **ua** in the same order as the matrix constraints were defined by nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc). The lower triangle of each is stored in the packed column order (see Section 3.3.2 in the f07 Chapter Introduction). For example, if there are four matrix constraints of dimensions 7, 3, 1, 1, the dimension of array **ua** should be 36. The first 28 elements $(d_1 \times (d_1 + 1)/2)$ belong to the packed lower triangle of U_1, followed by six elements of U_2 and one element for each U_3 and U_4. See for example Section 10 in nag_opt_sdp_read_sdpa (e04rdc).

3.2 Approximation of the Lagrangian Multipliers

By the nature of the algorithm, all inequality Lagrangian multiplier u, U are always kept positive during the computational process. This applies even to Lagrangian multipliers of inactive constraints at the solution. They will only be close to zero although they would normally be equal to zero exactly. This is one of the major differences between results from solvers based on the active set method (such as nag_opt_sparse_convex_qp_solve (e04nqc)) and others, such as, nag_opt_handle_solve_pennon (e04svc) or interior point methods. As a consequence, the initial estimate of the multipliers (if provided) might be adjusted by the solver to be sufficiently positive, also the estimates returned during the intermediate exits might only be a very crude approximation to their final values as they do not satisfy all the Karush–Kuhn–Tucker (KKT) conditions.

Another difference is that nag_opt_sparse_convex_qp_solve (e04nqc) merges multipliers for both lower and upper inequality into one element whose sign determines the inequality because there can be at most one active constraint and multiplier for the inactive is exact zero. Negative multipliers are associated with the upper bounds and positive with the lower bounds. On the other hand, E04SVF works with both multipliers at the same time so they are returned in two elements, one for the lower bound, the other for the upper bound (see Section 3.1). An equivalent result can be achieved by subtracting the upper bound multiplier from the lower one. This holds even when equalities are interpreted as two inequalities (see optional parameter **Transform Constraints**).

4 References

Ben–Tal A and Zibulevsky M (1997) Penalty/barrier multiplier methods for convex programming problems *SIAM Journal on Optimization* **7** 347–366

Fujisawa K, Kojima M, Nakata K (1997) Exploiting sparsity in primal-dual interior-point method for semidefinite programming *Math. Programming* **79** 235–253

Hogg J D and Scott J A (2011) HSL MA97: a bit-compatible multifrontal code for sparse symmetric systems *RAL Technical Report. RAL-TR-2011-024*

Kocõvara M and Stingl M (2003) PENNON – a code for convex nonlinear and semidefinite programming *Optimization Methods and Software* **18(3)** 317–333

Kocõvara M and Stingl M (2007) On the solution of large-scale SDP problems by the modified barrier method using iterative solvers *Math. Programming (Series B)* **109(2–3)** 413–444

Mittelmann H D (2003) An independent benchmarking of SDP and SOCP solvers *Math. Programming* **95** 407–430

Stingl M (2006) *On the Solution of Nonlinear Semidefinite Programs by Augmented Lagrangian Methods, PhD thesis* Institute of Applied Mathematics II, Friedrich–Alexander University of Erlangen–Nuremberg

5 Arguments

1: **handle** – void * *Input*

On entry: the handle to the problem. It needs to be initialized by nag_opt_handle_init (e04rac) and the problem formulated by some of the functions nag_opt_handle_set_linobj (e04rec), nag_opt_handle_set_quadobj (e04rfc), nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc), nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc). It **must not** be changed between the calls.

2: **nvar** – Integer *Input*

On entry: n, the number of decision variables x in the problem. It must be unchanged from the value set during the initialization of the handle by nag_opt_handle_init (e04rac).

3: **x**[**nvar**] – double *Input/Output*

Note: intermediate stops take place only if **Monitor Frequency** > 0.

On entry: if **Initial X** $=$ USER (the default), x^0, the initial estimate of the variables x, otherwise **x** need not be set.

On intermediate exit: the value of the variables x at the end of the current outer iteration.

On intermediate re-entry: the input is ignored.

On final exit: the final value of the variables x.

4: **nnzu** – Integer *Input*

On entry: the dimension of array **u**.

If **nnzu** $= 0$, **u** will not be referenced; otherwise it needs to match the dimension of constraints defined by nag_opt_handle_set_simplebounds (e04rhc) and nag_opt_handle_set_linconstr (e04rjc) as explained in Section 3.1.

Constraint: **nnzu** ≥ 0.

5: **u**[**nnzu**] – double *Input/Output*

Note: intermediate stops take place only if **Monitor Frequency** > 0.

If **nnzu** > 0, **u** holds Lagrangian multipliers (dual variables) for (standard) constraints, i.e., simple bounds defined by nag_opt_handle_set_simplebounds (e04rhc) and a set of m_B linear constraints defined by nag_opt_handle_set_linconstr (e04rjc). Either their initial estimates, intermediate approximations or final values, see Section 3.1.

If **nnzu** $= 0$, **u** will not be referenced and may be **NULL**.

On entry: if **Initial U** $=$ USER (the default is AUTOMATIC), u^0, the initial estimate of the Lagrangian multipliers u, otherwise **u** need not be set.

On intermediate exit: the estimate of the multipliers u at the end of the current outer iteration.

On intermediate re-entry: the input is ignored.

On exit: the final value of multipliers u.

6: **nnzuc** – Integer *Input*

On entry: the dimension of array **uc**. If **nnzuc** $= 0$, **uc** will not be referenced. This argument is reserved for future releases of the NAG C Library which will allow definition of second order cone constraints. It needs to be set to 0 at the moment.

Constraint: **nnzuc** $= 0$.

7: **uc**[**nnzuc**] – double *Input/Output*

uc is reserved for future releases of the NAG C Library which will allow definition of second order cone constraints. It is not referenced at the moment and may be **NULL**.

8: **nnzua** – Integer *Input*

On entry: the dimension of array **ua**. If **nnzua** $= 0$, **ua** will not be referenced; otherwise it needs to match the total number of nonzeros in all matrix Lagrangian multipliers (constraints defined by nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc)) as explained in Section 3.1.

Constraint: **nnzua** ≥ 0.

9: **ua**[**nnzua**] – double *Input/Output*

Note: intermediate stops take place only if **Monitor Frequency** > 0.

If **nnzua** > 0, **ua** holds the Lagrangian multipliers for matrix constraints defined by nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc), see Section 3.1.

If **nnzua** $= 0$, **ua** will not be referenced and may be **NULL**.

On entry: if **Initial U** $=$ USER (the default is AUTOMATIC), U^0, the initial estimate of the matrix Lagrangian multipliers U, otherwise **ua** need not be set.

On intermediate exit: the estimate of the matrix multipliers U at the end of the outer iteration.

On intermediate re-entry: the input is ignored.

On final exit: the final estimate of the multipliers U.

10: **rinfo**[**32**] – double*Output* error measures and various indicators (see Section 11 for details) at the end of the current (or final) outer iteration as given in the table below:

0	objective function value $f(x)$
1	optimality (12)
2	feasibility (13)
3	complementarity (14)
4	minimum penalty
5	relative precision (11)
6	relative duality gap (10)
7	precision $\left\|f\left(x^{\ell}\right) - f\left(x^{\ell+1}\right)\right\|$
8	duality gap
9	minimum penalty for (standard) inequalities p
10	minimum penalty for matrix inequalities P
11	feasibility of equality constraints
12	feasibility of (standard) inequalities
13	feasibility of matrix inequalities
14	complementarity of equality constraints
15	complementarity of (standard) inequalities
16	complementarity of matrix inequalities
17–22	DIMACS error measures (16) (only if turned on by **DIMACS Measures**)
23–31	reserved for future use

11: **stats**[**32**] – double *Output*

On intermediate or final exit: solver statistics at the end of the current (or final) outer iteration as given in the table below. Note that time statistics is provided only if **Stats Time** is set (the default is NO), the measured time is returned in seconds.

0	number of the outer iterations
1	total number of the inner iterations
2	total number of the linesearch steps
3	number of evaluations of the augmented Lagrangian $F()$, (see (8))
4	number of evaluations of $\nabla F()$
5	number of evaluations of $\nabla^2 F()$
6	reserved for future use
7	total running time of the solver

8	total running time of the solver without evaluations of the user's functions and monitoring stops
9	time spent in the inner iterations
10	time spent in Lagrangian multipliers updates
11	time spent in penalty parameters updates
12	time spent in matrix feasibility computation
13	time of evaluations of $F()$
14	time of evaluations of $\nabla F()$
15	time of evaluations of $\nabla^2 F()$
16	time of factorizations of the Newton system
17	time of factorizations of the matrix constraints
18–31	reserved for future use

12: **inform** – Integer * *Input/Output*

Note: intermediate stops take place only if **Monitor Frequency** > 0.

On initial entry: no effect.

On intermediate exit: **inform** $= 1$.

On intermediate re-entry: if set to 0, solving the current problem is terminated and the function returns **fail.code** = NE_USER_STOP; otherwise the routine continues.

On final exit: **inform** $= 0$.

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ALREADY_DEFINED

A different solver from the suite has already been used.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_DIM_MATCH

On entry, **nnzu** = ⟨*value*⟩.
nnzu does not match the size of the Lagrangian multipliers for (standard) constraints.
The correct value is 0 for no (standard) constraints.

On entry, **nnzu** = ⟨*value*⟩.
nnzu does not match the size of the Lagrangian multipliers for (standard) constraints.
The correct value is either 0 or ⟨*value*⟩.

On entry, **nnzua** = $\langle value \rangle$.
nnzua does not match the size of the Lagrangian multipliers for matrix constraints.
The correct value is 0 for no matrix constraints.

On entry, **nnzua** = $\langle value \rangle$.
nnzua does not match the size of the Lagrangian multipliers for matrix constraints.
The correct value is either 0 or $\langle value \rangle$.

On entry, **nnzuc** = $\langle value \rangle$.
nnzuc does not match the size of the Lagrangian multipliers for second order cone constraints.
The correct value is 0 for no such constraints.

NE_FAILED_START

The current starting point is unusable.

The starting point x^0, either provided by the user (if **Initial X** = *USER, the default) or the automatic estimate (if* **Initial X** = *AUTOMATIC), must not be extremely infeasible in the matrix constraints (infeasibility of order 10^6 and higher) and all the functions used in the problem formulation must be evaluatable.*

In the unlikely case this error is triggered, it is necessary to provide a better estimate of the initial values.

NE_HANDLE

The supplied **handle** does not define a valid handle to the data structure for the NAG optimization modelling suite. It has not been initialized by nag_opt_handle_init (e04rac) or it has been corrupted.

NE_INFEASIBLE

The problem was found to be infeasible during preprocessing.

*One or more of the constraints (or its part after preprocessing) violates the constraints by more than ϵ_{feas} (***Stop Tolerance Feasibility***).*

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_MAYBE_INFEASIBLE

The problem seems to be infeasible, the algorithm was stopped.

Whilst the algorithm cannot definitively detect that the problem is infeasible, several indirect indicators suggest that it might be the case.

NE_MAYBE_UNBOUNDED

The problem seems to be unbounded, the algorithm was stopped.

Whilst the algorithm cannot definitively detect that the problem is unbounded, several indirect indicators (such as a rapid decrease in the objective function and a lack of convergence in the inner subproblem) suggest that this might be the case. A good scaling of the objective function is always highly recommended to avoid situations when unusual behavior triggers falsely this error exit.

NE_NO_IMPROVEMENT

Unable to make progress, the algorithm was stopped.

This error is returned if the solver cannot decrease the duality gap over a range of iterations. This can be due to the scaling of the problem or the problem might be close to primal or dual infeasibility.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_REF_MATCH

On entry, **nvar** = $\langle value \rangle$, expected value = $\langle value \rangle$.
Constraint: **nvar** must match the value given during initialization of **handle**.

NE_SETUP_ERROR

This solver does not support general nonlinear objective and constraints.

NE_SUBPROBLEM

The inner subproblem could not be solved to the required accuracy.
Inner iteration limit has been reached.

The inner subproblem could not be solved to the required accuracy.
Limited progress in the inner subproblem triggered a stop (heuristic inner stop criteria).

The inner subproblem could not be solved to the required accuracy.
Line search or another internal component failed.

A problem with the convergence of the inner subproblem is typically a sign of numerical difficulties of the whole algorithm. The inner subproblem might be stopped before reaching the required accuracy because of the **Inner Iteration Limit**, *a heuristic detected no progress in the inner iterations (if* **Inner Stop Criteria** = *HEURISTIC, default) or if an internal component failed (for example, line search was unable to find a suitable step). The algorithm tries to recover, however, it might give up after several attempts with one of these error messages. If it occurs in the very early iterations, consider increasing* **Inner Stop Tolerance** *and possibly* **Init Value P** *or* **Init Value Pmat** *which should ease the first iterations. An occurrence in later iterations indicates numerical difficulties typically due to scaling and/or ill-conditioning or the problem is close to infeasible. Reducing the tolerance on the stopping criteria or increasing* **P Update Speed** *might be of limited help.*

NE_TOO_MANY_ITER

Outer iteration limit has been reached.
The requested accuracy is not achieved.

If **Outer Iteration Limit** *is left to the default, this error indicates numerical difficulties. Consider whether the stopping tolerances (***Stop Tolerance 1**, **Stop Tolerance 2**, **Stop Tolerance Feasibility***) are set too low or optional parameters affecting the behaviour of the penalty updates (***P Update Speed**, **P Min** *or* **Pmat Min***) have been modified inadvisedly. The iteration log should reveal more about the misbehaviour. Providing a different starting point might be of help in certain situations.*

NE_UNBOUNDED

The problem was found unbounded during preprocessing.

The objective function consists of an unrestricted ray and thus the problem does not have a solution.

NE_USER_STOP

User requested termination during a monitoring step.

NW_NOT_CONVERGED

The algorithm converged to a suboptimal solution.
The full accuracy was not achieved. The solution should still be usable.

This error may be reported only if **Stop Criteria** $=$ *SOFT (default). The solver predicted that it is unable to reach a better estimate of the solution. However, the error measures indicate that the point is a reasonable approximation. Typically, only the norm of the gradient of the Lagrangian (optimality) does not fully satisfy the requested tolerance whereas the others are well below the tolerance.*

Setting **Stop Criteria** $=$ *STRICT will disallow this error but it is unlikely that the algorithm would reach a better solution.*

7 Accuracy

The accuracy of the solution is driven by optional parameters **Stop Tolerance 1**, **Stop Tolerance 2**, **Stop Tolerance Feasibility** and **Stop Criteria** and in certain cases **DIMACS Measures**.

If **fail.code** $=$ NE_NOERROR on the final exit, the returned point satisfies Karush–Kuhn–Tucker (KKT) conditions to the requested accuracy (under the default settings close to $\sqrt{\epsilon}$) and thus it is a good estimate of a local solution. If **fail.code** $=$ NW_NOT_CONVERGED, some of the convergence conditions were not fully satisfied but the point still seems to be a reasonable estimate and should be usable. Please refer to Section 11.2 and the description of the particular options.

8 Parallelism and Performance

nag_opt_handle_solve_pennon (e04svc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_opt_handle_solve_pennon (e04svc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

9.1 Description of the Printed Output

The solver can print information to give an overview of the problem and of the progress of the computation. The output may be send to two independent streams (files) which are set by optional parameters **Print File** and **Monitoring File**. Optional parameters **Print Level**, **Print Options** and **Monitoring Level** determine the exposed level of detail. This allows, for example, to generate a detailed log in a file while the condensed information is displayed on the screen.

By default (**Print File** $= 6$, **Print Level** $= 2$), four sections are printed to the standard output: a header, a list of options, an iteration log and a summary.

Header

The header contains statistics about the size of the problem as represented internally, i.e., it reflects any changes imposed by preprocessing and problem transformations (see, for example, **Presolve Block Detect** and **Transform Constraints**). The header may look like:

```
E04SV, NLP-SDP Solver (Pennon)
----------------------------------------------------------------------------
 Number of variables           2                [eliminated            0]
                           simple  linear  nonlin
 (Standard) inequalities       3       0       0
 (Standard) equalities                 0       0
 Matrix inequalities                   1       1 [dense    2, sparse    0]
                                                 [max dimension        3]
```

It shows the total number of variables and how many of them were eliminated (e.g., fixed by a simple equality). A constraint with both a lower and an upper bound counts as 2 inequalities. Simple bounds are set by nag_opt_handle_set_simplebounds (e04rhc), matrix inequalities by nag_opt_handle_set_lin matineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc) and standard equalities and inequalities by nag_opt_handle_set_linconstr (e04rjc). Note that matrix constraints of dimension 1 are extracted and treated as (standard) inequalities as well. The header report concludes with the number of matrix constraints factorized as dense and sparse matrices, together with the largest dimension of the matrix inequalities.

Optional parameters list

The list shows all options of the solver, each displayed on one line. The line contains the option name, its current value and an indicator for how it was set. The options left at their defaults are noted by (d), the ones set by the user are noted by (U) and the options reset by the solver by (S). The solver will automatically set options which are set to AUTO or options which are not possible to satisfy in the given context (e.g., requesting **DIMACS Measures** for a nonlinear problem). Note that the output format is compatible with the file format expected by nag_opt_handle_opt_set_file (e04zpc). The output might look as follows:

```
Outer Iteration Limit         =               20 * U
 Stop Tolerance 1              =      1.00000E-06 * d
 Stop Tolerance 2              =      1.00000E-07 * d
 Hessian Density               =            Dense * S
```

Iteration log

If **Print Level** $= 2$, the status of each major iteration is condensed to one line. The line shows the major iteration number (0 represents the starting point), the current objective value, KKT measures (optimality, feasibility and complementarity), minimal penalty and the number of inner iterations performed. Note that all these values are also available in **rinfo**$[0], \ldots,$ **rinfo**$[4]$ and **stats**$[0]$. The output might look as follows:

```
 ------------------------------------------------------------------
  it |  objective |  optim  |   feas  |  compl  | pen min  | inner
 ------------------------------------------------------------------
   0   0.00000E+00  7.34E+00  1.23E-01  4.41E+01  1.00E+00   0
   1  -3.01998E-01  2.54E-03  0.00E+00  1.89E+00  1.00E+00   6
   2  -2.53008E+00  1.06E-03  1.30E-01  3.22E-01  3.17E-01   8
   3  -2.08172E+00  6.52E-03  1.85E-02  4.54E-02  1.01E-01   7
   4  -2.01060E+00  6.47E-03  4.10E-03  1.02E-02  3.19E-02   3
```

Occasionally, a one letter flag is printed at the end of the line indicating that the inner subproblem was not solved to the required accuracy. The possibilities are `M` for maximum number of inner iterations, `L` for difficulties in the line search and `!` when a heuristic stop took place. Repeated troubles in the subproblems may lead to **fail.code** = NE_SUBPROBLEM. The output below had **Inner Iteration Limit** $= 5$ which was not enough in the first subproblem (first outer iteration).

```
 ------------------------------------------------------------------
  it |  objective |  optim  |   feas  |  compl  | pen min | inner
 ------------------------------------------------------------------
   0  0.00000E+00  1.46E+03  5.01E+01  1.46E+03  6.40E+01   0
   1  3.78981E+02  3.86E+01  0.00E+00  1.21E+04  6.40E+01   5 M
   2  9.11724E+02  1.46E-02  0.00E+00  9.24E+02  4.45E+01   5
```

All KKT measures should normally converge to zero as the algorithm progresses and once the requested accuracy (**Stop Tolerance 2**) is achieved, the solver stops. However, the convergence is not necessarilly monotonic. The penalty parameters are decreased each major iteration which should improve overall the feasibility of the problem. This also increases the ill-conditioning which might lead to a higher number of inner iterations. A very high number of inner iterations usually signals numerical difficulties. See Section 11 for the algorithmic details.

If **Print Level** > 2, each major iteration produces significantly more detailed output comprising detailed error measures and output from every inner iteration. The output is self-explanatory so is not featured here in detail.

Summary

Once the solver finishes, a detailed summary is produced. An example is shown below:

```
 --------------------------------------------------------------
 Status: converged, an optimal solution found
 --------------------------------------------------------------
 Final objective value                2.300000E+01
 Relative precision                   5.873755E-09
 Optimality                           1.756062E-10
 Feasibility                          9.048738E-08
 Complementarity                      1.855566E-08
 DIMACS error 1                       8.780308E-11
 DIMACS error 2                       0.000000E+00
 DIMACS error 3                       0.000000E+00
 DIMACS error 4                       4.524369E-08
 DIMACS error 5                       4.065998E-10
 DIMACS error 6                       3.948012E-10
 Iteration counts
   Outer iterations                             13
   Inner iterations                             82
   Linesearch steps                             95
 Evaluation counts
   Augm. Lagr. values                           96
   Augm. Lagr. gradient                         96
   Augm. Lagr. hessian                          82
 Timing
   Total time                    0 h  0 min  3 sec
     Evaluations + monitoring            0.04 sec
     Solver itself                       3.09 sec
   Inner minimization step               2.72 sec   ( 87.1%)
     Augm. Lagr. value                   0.28 sec   (  9.0%)
     Augm. Lagr. gradient                0.67 sec   ( 21.6%)
     Augm. Lagr. hessian                 1.11 sec   ( 35.4%)
     system matr. factor.                0.64 sec   ( 20.5%)
     const. matr. factor.                0.40 sec   ( 12.8%)
   Multiplier update                     0.01 sec   (  0.3%)
   Penalty update                        0.02 sec   (  0.5%)
   Feasibility check                     0.15 sec   (  4.7%)
 --------------------------------------------------------------
```

It starts with the status line of the overall result which matches the **fail** value. It is followed by the final objective value and the error measures (including **DIMACS Measures** if turned on). Iteration counters, numbers of evaluations of the Augmented Lagrangian function and timing of the routine conclude the section. The timing of the algorithm is displayed only if **Stats Time** is set.

10 Example

Semidefinite Programming has many applications in several fields of mathematics, such as, combinatorial optimization, finance, statistics, control theory or structural optimization. However, these applications seldom come in the form of (2) or (3). Usually a reformulation is needed or even a relaxation is employed to achieve the desired formulation. This is also the case of the LovÄsz ϑ function computed in this example. See also Section 10 in nag_opt_handle_init (e04rac) for links to further examples in the suite.

The LovÄsz ϑ function (or also called ϑ number) of an undirected graph $G = (V, E)$ is an important quantity in combinatorial optimization. It gives an upper bound to Shannon capacity of the graph G and is also related to the clique number and the chromatic number of the complement of G which are NP-hard problems.

The ϑ function can be expressed in various ways, here we use the following:

$$\vartheta(G) = \text{minimize}\left\{\lambda_{\max}(H) \mid H \in \mathbb{S}^n, \quad s_{ij} = 1 \text{ if } i = j \text{ or if } ij \notin E\right\}$$

where $n = |V|$ and $\mathbb{S}^n$ denotes the space of real symmetric n by n matrices. This eigenvalue optimization problem is easy to reformulate as an SDP problem by introducing an artificial variable t as follows:

$$\begin{array}{ll} \underset{t,H}{\text{minimize}} & t \\ \text{subject to} & H \preceq tI \\ & H \in \mathbb{S}^n, \quad s_{ij} = 1 \text{ if } i = j \text{ or if } ij \notin E. \end{array}$$

Finally, this can be written as (2)) which is formulated in the example:

$$\begin{array}{ll} \underset{t,x}{\text{minimize}} & t \\ \text{subject to} & tI + \sum\limits_{ij \in E} x_{ij} E_{ij} - J \succeq 0 \end{array}$$

where J is a matrix of all ones and E_{ij} is a matrix of all zeros except (i,j) and (j,i).

The example also demonstrates how to set the optional parameters and how to retrieve them.

The data file stores the Petersen graph whose ϑ is 4.

10.1 Program Text

```
/* nag_opt_handle_solve_pennon (e04svc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Compute Lovasz theta number of the given graph G on the input
 * via semidefinite programming as
 *   min {lambda_max(H) | H is nv x nv symmetric matrix where
 *                         h_ij=1 if ij is not an edge or if i==j}
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

int main(void)
{
  Integer exit_status = 0;
  Integer i, idblk, idx, inform, ivalue, j, maxe, ne, nnzasum, nnzu,
         nnzua, nnzuc, nv, nvar;
  double rvalue, c[1], rinfo[32], stats[32], *a = 0, *x = 0;
  Integer idxc[1], *icola = 0, *irowa = 0, *nnza = 0, *va = 0, *vb = 0;
  char cvalue[41];
  void *handle = 0;
  /* Nag Types */
  NagError fail;
  Nag_VariableType optype;

  printf("nag_opt_handle_solve_pennon (e04svc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file. */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read in the number of vertices and edges of the graph. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &nv);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &nv);
```

```
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &ne);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &ne);
#endif

  if (!(va = NAG_ALLOC(ne, Integer)) || !(vb = NAG_ALLOC(ne, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Read in edges of the graph, accept only 1<=i<j<=nv. */
  maxe = ne;
  ne = 0;
  for (idx = 0; idx < maxe; idx++) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &i, &j);
#else
    scanf("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &i, &j);
#endif
    if (i >= 1 && i < j && j <= nv) {
      va[ne] = i;
      vb[ne] = j;
      ne++;
    }
  }

  /* Number of variables (same as edges in the graph plus one). */
  nvar = ne + 1;

  /* nag_opt_handle_init (e04rac).
   * Initialize an empty problem handle with NVAR variables. */
  nag_opt_handle_init(&handle, nvar, NAGERR_DEFAULT);

  idxc[0] = 1;
  c[0] = 1.0;
  /* nag_opt_handle_set_quadobj (e04rfc).
   * Add the quadratic objective to the handle. */
  nag_opt_handle_set_quadobj(handle, 1, idxc, c, 0, NULL, NULL, NULL,
                             NAGERR_DEFAULT);

  /* Generate matrix constraint as:
   *   sum_{ij is edge in G} x_ij*E_ij + t*I - J >=0
   * where J is the all-ones matrix. Its dimension is the same
   * as the number of vertices. */

  /* Total number of nonzeros: */
  nnzasum = ne + nv + (nv + 1) * nv / 2;
  if (!(nnza = NAG_ALLOC(nvar + 1, Integer)) ||
      !(irowa = NAG_ALLOC(nnzasum, Integer)) ||
      !(icola = NAG_ALLOC(nnzasum, Integer)) ||
      !(a = NAG_ALLOC(nnzasum, double)) || !(x = NAG_ALLOC(nvar, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* A_0 is all ones matrix. */
  idx = 0;
  nnza[0] = (nv + 1) * nv / 2;
  for (i = 1; i <= nv; i++)
    for (j = i; j <= nv; j++) {
      irowa[idx] = i;
      icola[idx] = j;
      a[idx] = 1.0;
      idx++;
    }
  /* A_1 is the identity. */
  nnza[1] = nv;
  for (i = 1; i <= nv; i++) {
    irowa[idx] = i;
    icola[idx] = i;
    a[idx] = 1.0;
    idx++;
  }
  /* A_2, A_3, ..., A_{ne+1} match the E_ij matrices. */
  for (i = 0; i < ne; i++) {
    nnza[2 + i] = 1;
    irowa[idx] = va[i];
    icola[idx] = vb[i];
    a[idx] = 1.0;
    idx++;
  }
  idblk = 0;

  /* nag_opt_handle_set_linconstr (e04rnc).
   * Add the linear matrix constraint to the problem formulation. */
  nag_opt_handle_set_linmatineq(handle, nvar, nv, nnza, nnzasum, irowa,
                                icola, a, 1, NULL, &idblk, NAGERR_DEFAULT);

  /* nag_opt_handle_opt_set (e04zmc).
   * Set optional arguments of the solver */
  nag_opt_handle_opt_set(handle, "Initial X = Automatic", NAGERR_DEFAULT);

  /* Pass the handle to the solver, we are not interested in
   * Lagrangian multipliers. */
  nnzu = 0;
  nnzuc = 0;
  nnzua = 0;
  INIT_FAIL(fail);
  /* nag_opt_handle_solve_pennon (e04svc). */
  nag_opt_handle_solve_pennon(handle, nvar, x, nnzu, NULL, nnzuc, NULL,
                              nnzua, NULL, rinfo, stats, &inform, &fail);

  if (fail.code == NE_NOERROR || fail.code == NW_NOT_CONVERGED) {
    /* Retrieve some of the settings by calling
     * nag_opt_handle_opt_get (e04znc). */
    nag_opt_handle_opt_get(handle, "Hessian Density", &ivalue, &rvalue,
                           cvalue, 40, &optype, NAGERR_DEFAULT);
    printf("The solver chose to use %s hessian", cvalue);
    nag_opt_handle_opt_get(handle, "Linesearch Mode", &ivalue, &rvalue,
                           cvalue, 40, &optype, NAGERR_DEFAULT);
    printf(" and %s as linesearch.\n", cvalue);
    printf("Lovasz theta number of the given graph is %7.2f.\n", rinfo[0]);
  }
  else {
    printf("Error from nag_opt_handle_solve_pennon (e04svc).\n%s\n",
           fail.message);
    exit_status = 1;
  }

END:

  /* nag_opt_handle_free (e04rzc).
   * Destroy the problem handle and deallocate all the memory. */
  if (handle)
    nag_opt_handle_free(&handle, NAGERR_DEFAULT);

  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(icola);
  NAG_FREE(irowa);
  NAG_FREE(nnza);
  NAG_FREE(va);
  NAG_FREE(vb);
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_handle_solve_pennon (e04svc) Example Program Data
10          : Number of vertices
15          : Number of edges
1 2         : List of edges, one per line,
2 3         : given as pairs i j of vertices (i<j)
3 4
4 5
1 5
1 6
2 7
3 8
4 9
5 10
6 8
6 9
7 9
7 10
8 10
```

10.3 Program Results

```
nag_opt_handle_solve_pennon (e04svc) Example Program Results

 E04SV, NLP-SDP Solver (Pennon)
 ------------------------------
 Number of variables           16                 [eliminated            0]
                           simple  linear  nonlin
 (Standard) inequalities        0       0       0
 (Standard) equalities                  0       0
 Matrix inequalities                    1       0 [dense    1, sparse    0]
                                                  [max dimension        10]

 Begin of Options
     Outer Iteration Limit          =                 100     * d
     Inner Iteration Limit          =                 100     * d
     Infinite Bound Size            =         1.00000E+20     * d
     Initial X                      =           Automatic     * U
     Initial U                      =           Automatic     * d
     Initial P                      =           Automatic     * d
     Hessian Density                =               Dense     * S
     Init Value P                   =         1.00000E+00     * d
     Init Value Pmat                =         1.00000E+00     * d
     Presolve Block Detect          =                 Yes     * d
     Print File                     =                   6     * d
     Print Level                    =                   2     * d
     Print Options                  =                 Yes     * d
     Monitoring File                =                  -1     * d
     Monitoring Level               =                   4     * d
     Monitor Frequency              =                   0     * d
     Stats Time                     =                  No     * d
     P Min                          =         1.05367E-08     * d
     Pmat Min                       =         1.05367E-08     * d
     U Update Restriction           =         5.00000E-01     * d
     Umat Update Restriction        =         3.00000E-01     * d
     Preference                     =               Speed     * d
     Transform Constraints          =                  No     * S
     Dimacs Measures                =               Check     * d
     Stop Criteria                  =                Soft     * d
     Stop Tolerance 1               =         1.00000E-06     * d
     Stop Tolerance 2               =         1.00000E-07     * d
     Stop Tolerance Feasibility     =         1.00000E-07     * d
     Linesearch Mode                =            Fullstep     * S
     Inner Stop Tolerance           =         1.00000E-02     * d
     Inner Stop Criteria            =           Heuristic     * d
     Task                           =            Minimize     * d
     P Update Speed                 =                  12     * d
 End of Options
 --------------------------------------------------------------
```

```
  it|  objective |  optim   |   feas   |  compl   | pen min |inner
 --------------------------------------------------------------
   0  0.00000E+00  4.71E+01  1.00E+01  4.81E+01  1.60E+01   0
   1  9.55399E+01  9.29E-03  0.00E+00  9.52E+01  1.60E+01   8
   2  3.93849E+01  1.16E-03  0.00E+00  3.81E+01  6.63E+00   5
   3  1.68392E+01  1.19E-02  0.00E+00  1.52E+01  2.75E+00   3
   4  8.50544E+00  7.32E-04  0.00E+00  5.78E+00  1.14E+00   4
   5  5.62254E+00  1.56E-02  0.00E+00  2.07E+00  4.72E-01   3
   6  4.63348E+00  7.66E-03  0.00E+00  7.33E-01  1.96E-01   4
   7  4.25322E+00  2.99E-03  0.00E+00  2.72E-01  8.11E-02   4
   8  4.10154E+00  2.41E-03  0.00E+00  1.05E-01  3.36E-02   4
   9  4.04076E+00  1.87E-03  0.00E+00  4.14E-02  1.39E-02   4
  10  4.01631E+00  6.25E-03  0.00E+00  1.65E-02  5.77E-03   5
  11  4.00656E+00  3.23E-03  0.00E+00  6.59E-03  2.39E-03   5
  12  4.00263E+00  2.89E-03  0.00E+00  2.64E-03  9.91E-04   5
  13  4.00106E+00  2.08E-03  0.00E+00  1.06E-03  4.11E-04   5
  14  4.00042E+00  1.53E-03  0.00E+00  4.25E-04  1.70E-04   5
 --------------------------------------------------------------
  it|  objective |  optim   |   feas   |  compl   | pen min |inner
 --------------------------------------------------------------
  15  4.00017E+00  1.30E-06  0.00E+00  1.70E-04  7.05E-05   6
  16  4.00007E+00  7.48E-07  0.00E+00  6.82E-05  2.92E-05   6
  17  4.00003E+00  3.20E-07  0.00E+00  2.73E-05  1.21E-05   6
  18  4.00001E+00  1.31E-07  0.00E+00  1.10E-05  5.02E-06   6
  19  4.00000E+00  5.15E-08  0.00E+00  4.39E-06  2.08E-06   6
  20  4.00000E+00  1.92E-08  0.00E+00  1.76E-06  8.62E-07   6
  21  4.00000E+00  7.06E-09  0.00E+00  7.05E-07  3.57E-07   6
  22  4.00000E+00  1.98E-09  0.00E+00  2.82E-07  1.48E-07   6
 --------------------------------------------------------------
 Status: converged, an optimal solution found
 --------------------------------------------------------------
 Final objective value               4.000000E+00
 Relative precision                  8.450361E-08
 Optimality                          1.983580E-09
 Feasibility                         0.000000E+00
 Complementarity                     2.822749E-07
 DIMACS error 1                      9.917898E-10
 DIMACS error 2                      0.000000E+00
 DIMACS error 3                      0.000000E+00
 DIMACS error 4                      0.000000E+00
 DIMACS error 5                      3.202984E-08
 DIMACS error 6                      3.136387E-08
 Iteration counts
   Outer iterations                            22
   Inner iterations                           112
   Linesearch steps                           308
 Evaluation counts
   Augm. Lagr. values                         135
   Augm. Lagr. gradient                       135
   Augm. Lagr. hessian                        112
 --------------------------------------------------------------
The solver chose to use DENSE hessian and FULLSTEP as linesearch.
Lovasz theta number of the given graph is    4.00.
```

11 Algorithmic Details

This section contains a description of the algorithm used in nag_opt_handle_solve_pennon (e04svc) which is based on the implementation of the code called Pennon. For further details, see Kocõvara and Stingl (2003), Stingl (2006) and Kocõvara and Stingl (2007).

For simplicity, we will use the following problem formulation; its connection to (SDP) and (BMI-SDP) is easy to see:

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) & \\ \text{subject to} & g_k(x)\geq 0, & k=1,2,\dots,m_g \\ & h_k(x)=0, & k=1,2,\dots,m_h \\ & \mathcal{A}_k(x)\succeq 0, & k=1,2,\dots,m_A, \end{array} \tag{4}$$

where f, g_k, h_k are $\mathcal{C}^2$ functions from $\mathbb{R}^n$ to $\mathbb{R}$ and $\mathcal{A}_k$ is a $\mathcal{C}^2$ matrix function from $\mathbb{R}^n$ to $\mathbb{S}^{m_k}$. Here $\mathbb{S}^m$ denotes the space of real symmetric matrices $m \times m$ and $S \in \mathbb{S}^m$, $S \succeq 0$ stands for a constraint on eigenvalues of S, namely the matrix S should be positive semidefinite. Furthermore, we define the inner product on $\mathbb{S}^m$ by $\langle A, B \rangle_{\mathbb{S}^m} = \mathrm{trace}(AB)$. The index $\mathbb{S}^m$ will be omitted whenever the dimension is clear from the context. Finally, for $\Phi : \mathbb{S}^m \to \mathbb{S}^m$ and $X, Y \in \mathbb{S}^m$, $D\Phi(X; Y)$ denotes the directional derivative of Φ with respect to X in direction Y.

11.1 Overview

The algorithm is based on a (generalized) augmented Lagrangian approach and on a suitable choice of smooth penalty/barrier functions $\varphi_g : \mathbb{R} \to \mathbb{R}$ for (standard) inequality constraints and $\varphi_A : \mathbb{R} \to \mathbb{R}$ for constraints on matrix eigenvalues. By means of φ_A we define a penalty/barrier function for matrix inequalities as follows.

Let $A \in \mathbb{S}^m$ have an eigenvalue decomposition $A = S^{\mathrm{T}} \Lambda S$ where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)^{\mathrm{T}}$. We define matrix function $\Phi_P : \mathbb{S}^m \to \mathbb{S}^m$ for $P > 0$ as

$$\Phi_P : A \longmapsto S^{\mathrm{T}} \begin{pmatrix} P\varphi_A\left(\frac{\lambda_1}{P}\right) & 0 & \cdots & 0 \\ 0 & P\varphi_A\left(\frac{\lambda_2}{P}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P\varphi_A\left(\frac{\lambda_m}{P}\right) \end{pmatrix} S. \tag{5}$$

Both φ_g and φ_A satisfy a number of assumptions (see Kocõvara and Stingl (2003)) guaranteeing, in particular, that for any p, $P > 0$

$$\begin{array}{lcll} g_k(x) \geq 0 & \Leftrightarrow & p\varphi_g(g_k(x)/p) \geq 0, & k = 1, 2, \ldots, m_g, \\ \mathcal{A}_k(x) \succeq 0 & \Leftrightarrow & \Phi_P(\mathcal{A}_k(x)) \succeq 0, & k = 1, 2, \ldots, m_A. \end{array} \tag{6}$$

Further in the text, we use simplified notation $\varphi_p(\cdot) = p\varphi_g(\cdot/p)$.

Thus for any p, $P > 0$, problem (4) has the same solution as the following augmented problem

$$\begin{array}{lll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) & \\ \text{subject to} & \varphi_p(g_k(x)) \geq 0, & k = 1, 2, \ldots, m_g \\ & h_k(x) = 0, & k = 1, 2, \ldots, m_h \\ & \Phi_P(\mathcal{A}_k(x)) \succeq 0, & k = 1, 2, \ldots, m_A. \end{array} \tag{7}$$

The Lagrangian of (7) can be viewed as a (generalized) augmented Lagrangian of (4):

$$\begin{aligned} F(x, u, v, U, p, P) \quad = \quad & f(x) - \sum_{k=1}^{m_g} u_k \varphi_p(g_k(x)) \\ & + \sum_{k=1}^{m_h} v_k h_k(x) \\ & - \sum_{k=1}^{m_A} \langle U_k, \Phi_P(\mathcal{A}_k(x)) \rangle; \end{aligned} \tag{8}$$

where $u \in \mathbb{R}^{m_g}$, $v \in \mathbb{R}^{m_h}$ and $U = (U_1, \ldots, U_{m_A})$, $U_k \in \mathbb{S}^{p_k}$, $k = 1, \ldots, m_A$ are Lagrange multipliers associated with the (standard) inequalities and equalities and the matrix inequality constraints, respectively.

The algorithm combines ideas of the (exterior) penalty and (interior) barrier methods with the augmented Lagrangian method, it can be defined as follows:

Algorithm 1 (Outer Loop) Let x^0, u^0, v^0 and U^0 be given. Let $p^0 > 0$, $P^0 > 0$, $\alpha^0 > 0$. For $\ell = 0, 1, \ldots$ repeat until a stopping criteria or maximum number of iterations is reached:

(i) Find $x^{\ell+1}$, $v^{\ell+1}$ satisfying

$$\begin{Vmatrix}\nabla_x F\left(x^{\ell+1}, u^\ell, v^{\ell+1}, U^\ell, p^\ell, P^\ell\right)\| \le \alpha^\ell \\ h\left(x^{\ell+1}\right)\| \le \alpha^\ell\end{Vmatrix} \tag{9}$$

(ii) Update Lagrangian multipliers

$$\begin{aligned} U_k^{\ell+1} &= D\Phi_P\left(\mathcal{A}_k\left(x^{\ell+1}\right); U_k^\ell\right), \qquad & k = 1, 2, \ldots, m_A \\ u_k^{\ell+1} &= u_k^\ell \varphi_g^{'}\left(g_k\left(x^{\ell+1}\right)/p^\ell\right), \qquad & k = 1, 2, \ldots, m_g \end{aligned}$$

(iii) Update penalty parameters and inner problem stopping criteria

$$p^{\ell+1} < p^\ell, \quad P^{\ell+1} < P^\ell, \quad \alpha^{\ell+1} \le \alpha^\ell.$$

Step (i) of Algorithm 1, further referred as the *inner* problem, is the most time-consuming and thus the choice of the solver for (9) is critical for the overall efficiency of the method. See Section 11.4 below.

The inequality Lagrangian multipliers update in step (ii) is motivated by the fact that if $x^{\ell+1}$, $v^{\ell+1}$ solve (9) exactly in iteration ℓ, we obtain

$$\nabla_x F\left(x^{\ell+1}, u^{\ell+1}, v^{\ell+1}, U^{\ell+1}, p^\ell, P^\ell\right) = 0.$$

Details can be found, for example, in Stingl (2006).

In practise, numerical studies showed that it is not advantageous to do the full updates of multipliers u, U. Firstly, big changes in the multipliers may lead to a large number of iterations in subsequent solution of (9) and, secondly, the multipliers might become ill-conditioned after a few steps and the algorithm suffers from numerical instabilities. To overcome these difficulties, a restricted update is performed instead.

New Lagrangian multipliers for (standard) inequalities $u_k^{\ell+1}$, for $k = 1, 2, \ldots, m_g$ are limited not to violate the following bound

$$\mu_g < \frac{u_k^{\ell+1}}{u_k^\ell} < \frac{1}{\mu_g}$$

for a given $0 < \mu_g < 1$ (see **U Update Restriction**).

A similar strategy is applied to the matrix multipliers $U_k^{\ell+1}$ as well. For $0 < \mu_A < 1$ (see **Umat Update Restriction**) set

$$U_k^{\text{new}} = U_k^{\ell+1} + \mu_A\left(U_k^\ell - U_k^{\ell+1}\right).$$

The penalty parameters p, P in step (iii) are updated by some constant factor dependent on the initial penalty parameters p^0, P^0 and **P Update Speed**. The update process is stopped when p_{min} and P_{min} are reached (see **P Min**, **Pmat Min**).

Additional details about the multiplier and penalty update strategies, as well as local and global convergence properties under standard assumptions can be found in an extensive study Stingl (2006).

11.2 Stopping Criteria

Algorithm 1 is stopped when all the stopping criteria are satisfied to the requested accuracy, these are:

$$\frac{\left|f\left(x^\ell\right) - F\left(x^\ell, u^\ell, v^\ell, U^\ell, p^\ell, P^\ell\right)\right|}{1 + \left|f\left(x^\ell\right)\right|} \le \epsilon_1, \qquad (\textit{relative duality gap}) \tag{10}$$

$$\frac{\left|f\left(x^\ell\right) - f\left(x^{\ell-1}\right)\right|}{1 + \left|f\left(x^\ell\right)\right|} \le \epsilon_1, \qquad (\textit{relative precision}) \tag{11}$$

and these based on Karush–Kuhn–Tucker (KKT) error measures, to keep the notation simple, formulation (4) is assumed and iteration index ℓ is dropped:

$$\left\| \nabla f(x) - \sum_{k=1}^{m_g} u_k \nabla g_k(x) + \sum_{k=1}^{m_h} v_k \nabla h_k(x) - \sum_{k=1}^{m_A} \left[\left\langle U_k, \frac{\partial}{\partial x_i} \mathcal{A}_k(x) \right\rangle \right]_{i=1,\dots,n} \right\| \leq \epsilon_2, \qquad (\mathit{optimality}) \tag{12}$$

$$g_k(x) \geq -\epsilon_{\text{feas}}, \quad |h_k(x)| \leq \epsilon_{\text{feas}}, \quad \mathcal{A}_k(x) \succeq -\epsilon_{\text{feas}} I \quad \text{for all } k, \qquad (\mathit{feasibility}) \tag{13}$$

$$|g_k(x) u_k| \leq \epsilon_2, \quad |h_k(x) v_k| \leq \epsilon_2, \quad |\langle \mathcal{A}_k(x), U_k \rangle| \leq \epsilon_2. \qquad (\mathit{complementarity}) \tag{14}$$

Here ϵ_1, ϵ_2, ϵ_{feas} may be set in the option settings as **Stop Tolerance 1**, **Stop Tolerance 2** and **Stop Tolerance Feasibility**, respectively.

Note that if **Task** $=$ FEASIBLE POINT, only the feasibility is taken into account.

There is an option for linear SDP problems to switch from stopping criteria based on the KKT conditions to **DIMACS Measures**, see Mittelmann (2003). This is the default choice. To keep the notation readable, these are defined here only for the following simpler formulation of linear SDP rather than (2):

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & c^{\mathrm{T}} x \\ \text{subject to} & \mathcal{A}(x) = \sum_{i=1}^{n} x_i A_i - A_0 \succeq 0. \end{array} \tag{15}$$

In this case the algorithm stops when:

$$\begin{aligned} \text{Derr}_1 &= \frac{\|\mathcal{A}^*(U) - c\|}{1 + \|c\|} \\ \text{Derr}_2 &= \max\left(0, \frac{-\lambda_{\min}(U)}{1 + \|c\|}\right) \\ \text{Derr}_4 &= \max\left(0, \frac{-\lambda_{\min}\left(\sum_{i=1}^{n} x_i A_i - A_0\right)}{1 + \|A_0\|}\right) \\ \text{Derr}_5 &= \frac{\langle A_0, U \rangle - c^{\mathrm{T}} x}{1 + |\langle A_0, U \rangle| + c^{\mathrm{T}} x} \\ \text{Derr}_6 &= \frac{\left\langle \sum_{i=1}^{n} x_i A_i - A_0, U \right\rangle}{1 + |\langle A_0, U \rangle| + |c^{\mathrm{T}} x|} \end{aligned} \tag{16}$$

where $\mathcal{A}^*(\cdot)$ denote the adjoint operator to $\mathcal{A}(\cdot)$, $[\mathcal{A}^*(U)]_i = \langle A_i, U \rangle$.

They can be viewed as a scaled version of the KKT conditions. Derr_1 represents the (scaled) norm of the gradient of the Lagrangian, Derr_2 and Derr_4 the dual and primal infeasibility, respectively, and Derr_5 and Derr_6 measure the duality gap and the complementary slackness. Note that in this solver $\text{Derr}_2 = 0$ by definition and Derr_3 is automaticaly zero because the formulation involves slack variables which are not used here.

11.3 Choice of penalty functions φ_g and φ_A

To treat the (standard) inequality constraints $g_k(x) \geq 0$, we use the penalty/barrier function proposed by Ben–Tal and Zibulevsky (1997):

$$\varphi_g(\tau) = \begin{cases} -\tau + \frac{1}{2}\tau^2 & \text{if } \tau \leq \bar{\tau} \\ -(1-\bar{\tau})^2 \log\left(\frac{1-2\bar{\tau}+\tau}{1-\bar{\tau}}\right) - \bar{\tau} + \frac{1}{2}\bar{\tau}^2 & \text{if } \tau > \bar{\tau}; \end{cases}$$

with default $\bar{\tau} = \frac{1}{2}$.

The choice of φ_A (and thus of Φ_P) is motivated by the complexity of the evaluation of Φ_P and its derivatives. If φ_A is defined as

$$\varphi_A(\tau) = \frac{1}{1+\tau} - 1,$$

it is possible to avoid the explicit eigenvalue decomposition in (5) as it can be seen in the formulae below (note that index k is omitted):

$$\begin{aligned} \Phi_P(\mathcal{A}(x)) &= P^2\mathcal{Z}(x) - PI \\ \frac{\partial}{\partial x_i}\Phi_P(\mathcal{A}(x)) &= -P^2\mathcal{Z}(x)\frac{\partial\mathcal{A}(x)}{\partial x_i}\mathcal{Z}(x) \\ \frac{\partial^2}{\partial x_i \partial x_j}\Phi_P(\mathcal{A}(x)) &= P^2\mathcal{Z}(x)\left(\frac{\partial\mathcal{A}(x)}{\partial x_i}\mathcal{Z}(x)\frac{\partial\mathcal{A}(x)}{\partial x_j} - \frac{\partial^2\mathcal{A}(x)}{\partial x_i\partial x_j} + \frac{\partial\mathcal{A}(x)}{\partial x_j}\mathcal{Z}(x)\frac{\partial\mathcal{A}(x)}{\partial x_i}\right)\mathcal{Z}(x) \end{aligned} \tag{17}$$

where

$$\mathcal{Z}(x) = (\mathcal{A}(x) + PI)^{-1}. \tag{18}$$

For details follow Kocõvara and Stingl (2003). Note that, in particular, formula (17) requires nontrivial computational resources even if careful handling of the sparsity of partial derivatives of $\mathcal{A}(x)$ is implemented. nag_opt_handle_solve_pennon (e04svc) uses a set of strategies described in Fujisawa *et al.* (1997) adapted for parallel computation.

11.4 Solution of the inner problem

This section describes solving of the inner problem (step (i) of Algorithm 1). We attempt to find an approximate solution of the following system (in x and v) up to the given precision α:

$$\begin{aligned} \nabla_x F(x,u,v,U,p,P) &= 0 \\ h(x) &= 0 \end{aligned} \tag{19}$$

where the penalty parameters p, P, as well as the Lagrangian multipliers u and U are fixed.

A linesearch SQP framework is used due to its desirable convergence properties. It can be stated as follows.

Algorithm 2 (Inner Loop) Let x^0, v^0 be given (typically as the solution from the previous outer iteration), p, P, u, U and $\alpha > 0$ fixed. For $\ell = 0, 1, \ldots$

(i) Find a descent direction d by solving

$$\begin{pmatrix} \nabla^2 F(x^\ell) & \nabla h(x^\ell) \\ \nabla h(x^\ell)^{\mathrm{T}} & 0 \end{pmatrix}\begin{pmatrix} d \\ d_v \end{pmatrix} = -\begin{pmatrix} \nabla F(x^\ell) \\ h(x^\ell) \end{pmatrix} \tag{20}$$

(ii) Find a suitable step length δ and set

$$\begin{aligned} x^{\ell+1} &= x^\ell + \delta d \\ v^{\ell+1} &= v^\ell + \delta d_v \end{aligned}$$

(iii) Stop if **Inner Iteration Limit** is reached or if

$$\begin{aligned} &\left\|\nabla_x F(x^{\ell+1}, u, v^{\ell+1}, U, p, P)\right\| \leq \alpha \\ &\left\|h(x^{\ell+1})\right\| \leq \alpha. \end{aligned}$$

System (20) is solved by the factorization routine MA97 (see Hogg and Scott (2011), in combination with an inertia correction strategy described in Stingl (2006). The step length selection is guided by **Linesearch Mode**.

If there are no equality constraints in the problem, the unconstrained minimization in Step (i) of Algorithm 1 simplifies to the modified Newton method with line-search (for details, see Kocõvara and Stingl (2003)). Alternatively, the equality constraints $h_k(x) = 0$ can be converted to two inequalities which would be treated with the remaining constraints (see **Transform Constraints**).

12 Optional Parameters

Several optional parameters in nag_opt_handle_solve_pennon (e04svc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_opt_handle_solve_pennon (e04svc) these optional parameters have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters.

The optional parameters can be changed by calling nag_opt_handle_opt_set (e04zmc) anytime between the initialization of the handle by nag_opt_handle_init (e04rac) and the call to the solver. Modification of the arguments during intermediate monitoring stops is not allowed. Once the solver finishes, the optional parameters can be altered again for the next solve.

If any options are set by the solver (typically those with the choice of AUTO), their value can be retrieved by nag_opt_handle_opt_get (e04znc). If the solver is called again, any such arguments are reset to their default values and the decision is made again.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 12.1.

Defaults
DIMACS Measures
Hessian Density
Infinite Bound Size
Initial P
Initial U
Initial X
Init Value P
Init Value Pmat
Inner Iteration Limit
Inner Stop Criteria
Inner Stop Tolerance
Linesearch Mode
List
Monitor Frequency
Monitoring File
Monitoring Level
Outer Iteration Limit
Pmat Min
P Min
Preference
Presolve Block Detect
Print File
Print Level

Print Options
P Update Speed
Stats Time
Stop Criteria
Stop Tolerance 1
Stop Tolerance 2
Stop Tolerance Feasibility
Task
Transform Constraints
Umat Update Restriction
U Update Restriction

12.1 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, where the minimum abbreviation of each keyword is underlined;

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

the default value, where the symbol ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

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

Keywords and character values are case and white space insensitive.

Defaults

This special keyword may be used to reset all optional parameters to their default values. Any argument value given with this keyword will be ignored.

DIMACS Measures a Default $=$ CHECK

If the problem is a linear semidefinite programming problem, this argument specifies if DIMACS error measures (see Section 11.2) should be computed and/or checked. In other cases, this option reverts to NO automatically.

Constraint: **DIMACS Measures** $=$ COMPUTE, CHECK or NO.

Hessian Density a Default $=$ AUTO

This optional parameter guides the solver on how the Hessian matrix of augmented Lagrangian $F(x,u,v,U,p,P)$ should be built. Option AUTO leaves the decision to the solver and it is the recommended option. Setting it to DENSE bypasses the autodetection and the Hessian is always built as a dense matrix. Option SPARSE instructs the solver to use a sparse storage and factorization of the matrix if possible.

Constraint: **Hessian Density** $=$ AUTO, DENSE or SPARSE

Infinite Bound Size r Default $= 10^{20}$

This defines the 'infinite' bound $bigbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $bigbnd$ will be regarded as $+\infty$ (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as $-\infty$). Note that a modification of this optional parameter does not influence constraints which have already been defined; only the constraints formulated after the change will be affected.

Constraint: **Infinite Bound Size** ≥ 1000.

Initial P a Default $=$ AUTOMATIC

This optional parameter defines the choice of the penalty optional parameters p^0, P^0, see Algorithm 1.

Initial P $=$ AUTOMATIC
: The penalty optional parameters are chosen automatically as set by optional parameter **Init Value P**, **Init Value Pmat** and subject to automatic scaling. Note that P^0 might be increased so that the penalty function $\Phi_P()$ is defined for all matrix constraints at the starting point.

Initial P $=$ KEEP PREVIOUS
: The penalty optional parameters are kept from the previous run of the solver if possible. If not, this options reverts to AUTOMATIC. Note that even if the matrix penalty optional parameters are the same as in the previous run, they are still subject to a possible increase so that the penalty function $\Phi_P()$ is well defined at the starting point.

Constraint: **Initial P** $=$ AUTOMATIC or KEEP PREVIOUS.

Initial U a Default $=$ AUTOMATIC

This argument guides the solver on which initial Lagrangian multipliers are to be used.

Initial U $=$ AUTOMATIC
: The Lagrangian multipliers are chosen automatically as set by automatic scaling.

Initial U $=$ USER
: The values of arrays **u** and **ua** (if provided) are used as the initial Lagrangian multipliers subject to automatic adjustments. If one or the other array is not provided, the choice for missing data is as in AUTOMATIC.

Initial U $=$ KEEP PREVIOUS
: The Lagrangian multipliers are kept from the previous run of the solver. If this option is set for the first run or optional parameters change the approach of the solver, the choice automatically reverts to AUTOMATIC. This might be useful if the solver is hot started, for example, to achieve higher precision of the solution.

Constraint: **Initial U** $=$ AUTOMATIC, USER or KEEP PREVIOUS.

Initial X a Default $=$ USER

This argument guides which starting point x^0 is to be used.

Initial X $=$ AUTOMATIC
: The starting point is chosen automatically so that it satisfies simple bounds on the variables or as a zero vector. Input of argument **x** is ignored.

Initial X $=$ USER
: Initial values of argument **x** are used as a starting point.

Constraint: **Initial X** $=$ AUTOMATIC or USER.

Init Value P r Default $= 1.0$

This argument defines the value p^0, the initial penalty optional parameter for (standard) inequalities. A low value of the penalty causes the solution of the inner problem to be closer to the feasible region and thus to the desirable result. However, it also increases ill-conditioning of the system. It is not advisable to set the penalty too low unless a good starting point is provided.

Constraint: $\sqrt[4]{\epsilon} \leq$ **Init Value P** $\leq 10^4$.

Init Value Pmat r Default $= 1.0$

The value of this option suggests P^0, the initial penalty optional parameter for matrix inequalities. It is similar to **Init Value P** (and the same advice applies), however, P^0 gets increased automatically if the matrix constraints are more infeasible than the actual penalty optional parameter.

Constraint: $\sqrt[4]{\epsilon} \leq$ **Init Value Pmat** $\leq 10^4$.

Inner Iteration Limit i Default $= 100$

The maximum number of the inner iterations (Newton steps) to be performed by Algorithm 2 in each outer iteration. Setting the option too low might lead to **fail.code** = NE_SUBPROBLEM. Values higher than 100 are unlikely to improve convergence.

Constraint: **Inner Iteration Limit** > 0.

Inner Stop Criteria a Default = HEURISTIC

The precision α for the solution of the inner subproblem is determined in Algorithm 1 and under typical circumstances Algorithm 2 is expected to reach this precision within the given **Inner Iteration Limit**. If any problems are detected and **Inner Stop Criteria** = HEURISTIC, Algorithm 2 is allowed to stop before reaching the requested precision or the **Inner Iteration Limit**. This usually saves many unfruitful iterations and the solver may recover in the following iterations. If you suspect that the heuristic problem detection is not suitable for your problem, setting **Inner Stop Criteria** = STRICT disallows such behaviour.

Constraint: **Inner Stop Criteria** = HEURISTIC or STRICT.

Inner Stop Tolerance r Default $= 10^{-2}$

This option sets the required precision α^0 for the first inner problem solved by Algorithm 2. The precison of the solution of the inner problem does not need to be very high in the first outer iterations and it is automatically adjusted through the outer iterations to reach the optimality limit ϵ_2 in the last one.

Setting α^0 too restrictive (too low) causes an increase of the number of inner iterations needed in the first outer iterations and might lead to **fail.code** = NE_SUBPROBLEM. In certain cases it might be helpful to use a more relaxed (higher) α^0 and increase **P Update Speed** which should reduce the number of inner iterations needed at the beginning of the computation in exchange for a possibly higher number of the outer iterations.

Constraint: $\epsilon <$ **Inner Stop Tolerance** $\leq 10^3$.

Linesearch Mode a Default = AUTO

This controls the step size selection in Algorithm 2. If **Linesearch Mode** = FULLSTEP (the default for linear problems), unit steps are taken where possible and the step shortening takes place only to avoid undefined regions for the matrix penalty function $\Phi_P()$ (see (17)). This may be used for linear problems but it is not recommended for any nonlinear ones. If **Linesearch Mode** = ARMIJO, Armijo backtracking linesearch is used instead which is a fairly basic linesearch. If **Linesearch Mode** = GOLDSTEIN, a cubic safe guarded linesearch based on Goldstein condition is employed, this is the recommended (and default) choice for nonlinear problems.

Constraint: **Linesearch Mode** = AUTO, FULLSTEP, ARMIJO or GOLDSTEIN.

List a Default = NO

This argument may be set to YES if you wish to turn on printing of each optional parameter specification as it is supplied.

Constraint: **List** = YES or NO

Monitor Frequency i Default $= 0$

If **Monitor Frequency** > 0, the solver returns to you at the end of every ith outer iteration. During these intermediate exits, the current point **x** and Lagrangian multipliers **u**, **ua** (if requested) are provided as well as the statistics and error measures (**rinfo**, **stats**). Argument **inform** helps to distinguish between intermediate and final exits and also allows immediate termination.

If **Monitor Frequency** $= 0$, the solver stops only once on the final point and no intermediate exits are made.

Constraint: **Monitor Frequency** ≥ 0.

Monitoring File i Default $= -1$

(See Section 2.3.1.1 in How to Use the NAG Library and its Documentation for further information on NAG data types.)

If $i \geq 0$, the Nag_FileID number (as returned from nag_open_file (x04acc)) for the secondary (monitoring) output. If set to -1, no secondary output is provided. The following information is output to the unit:

– a listing of the optional parameters;

– problem statistics, the iteration log and the final status as set by **Monitoring Level**.

Constraint: **Monitoring File** ≥ -1.

Monitoring Level i Default $= 4$

This argument sets the amount of information detail that will be printed by the solver to the secondary output. The meaning of the levels is the same as with **Print Level**.

Constraint: $0 \leq$ **Monitoring Level** ≤ 5.

Outer Iteration Limit i Default $= 100$

The maximum number of the outer iterations to be performed by Algorithm 1. If **Outer Iteration Limit** $= 0$, no iteration is performed, only quantities needed in the stopping criteria are computed and returned in **rinfo**. This might be useful in connection with **Initial X** $=$ USER and **Initial U** $=$ USER to check optimality of the given point. However, note that the rules for possible modifications of the starting point still apply, see **u** and **ua**. Setting the option too low might lead to **fail**.**code** $=$ NE_TOO_MANY_ITER.

Constraint: **Outer Iteration Limit** ≥ 0.

P Min r Default $= \sqrt{\epsilon}$

This controls $p_{\min}$, the lowest possible penalty value p used for (standard) inequalities. In general, very small values of the penalty optional parameters cause ill-conditioning which might lead to numerical difficulties. On the other hand, very high $p_{\min}$ prevents the algorithm from reaching the requested accuracy on the feasibility. Under normal circumstances, the default value is recommended.

Constraint: $\epsilon \leq$ **P Min** $\leq 10^{-2}$.

Pmat Min r Default $= \sqrt{\epsilon}$

This is an equivalent of **P Min** for the minimal matrix penalty optional parameter $P_{\min}$. The same advice applies.

Constraint: $\epsilon \leq$ **Pmat Min** $\leq 10^{-2}$.

Preference a Default $=$ SPEED

This option affects how contributions from the matrix constraints (17) to the system Hessian matrix are computed. The default option of **Preference** $=$ SPEED should be suitable in most cases. However, dealing with matrix constraints of a very high dimension may cause noticable memory overhead and switching to **Preference** $=$ MEMORY may be required.

Constraint: **Preference** $=$ SPEED or MEMORY.

Presolve Block Detect a Default $=$ YES

If **Presolve Block Detect** $=$ YES, the matrix constraints are checked during preprocessoring to determine if they can be split into smaller independent ones, thus speeding up the solver.

Constraint: **Presolve Block Detect** $=$ YES or NO.

Print File i Default $= 6$

(See Section 2.3.1.1 in How to Use the NAG Library and its Documentation for further information on NAG data types.)

If $i \geq 0$, the Nag_FileID number (as returned from nag_open_file (x04acc), `stdout` as the default) for the primary output of the solver. If **Print File** $= -1$, the primary output is completely turned off independently of other settings. The following information is output to the unit:

– a listing of optional parameters if set by **Print Options**;

– problem statistics, the iteration log and the final status from the solver as set by **Print Level**.

Constraint: **Print File** ≥ -1.

Print Level i Default $= 2$

This argument defines how detailed information should be printed by the solver to the primary output.

i	**Output**
0	No output from the solver
1	Only the final status and the objective value
2	Problem statistics, one line per outer iteration showing the progress of the solution, final status and statistics
3	As level 2 but detailed output of the outer iterations is provided and brief overview of the inner iterations
4, 5	As level 3 but details of the inner iterations are printed as well

Constraint: $0 \leq$ **Print Level** ≤ 5.

Print Options a Default $=$ YES

If **Print Options** $=$ YES, a listing of optional parameters will be printed to the primary output.

Constraint: **Print Options** $=$ YES or NO.

P Update Speed i Default $= 12$

This option affects the rate at which the penalty optional parameters p, P are updated (Algorithm 1, step (iii)) and thus indirectly influences the overall number of outer iterations. Its value can be interpretted as the typical number of outer iterations needed to get from the initial penalty values p^0, P^0 half-way to the $p_{\min}$ and $P_{\min}$. Values smaller than 3 causes a very agressive penalty update strategy which might lead to the increased number of inner iterations and possibly to numerical difficulties. On the other hand, values higher than 15 produce a relatively conservative approach which leads to a higher number of the outer iterations.

If the solver encounters difficulties on your problem, a higher value might help. If your problem is working fine, setting a lower value might increase the speed.

Constraint: $1 \leq$ **P Update Speed** ≤ 100.

Stats Time a Default $=$ NO

This argument turns on timings of various parts of the algorithm to give a better overview of where most of the time is spent. This might be helpful for a choice of different solving approaches. It is possible to choose between CPU and wall clock time. Choice YES is equivalent to wall clock.

Constraint: **Stats Time** $=$ YES, NO, CPU or WALL CLOCK.

Stop Criteria *a* Default = SOFT

If **Stop Criteria** = SOFT, the solver is allowed to stop prematurely with a suboptimal solution, **fail.code** = NW_NOT_CONVERGED, if it predicts that a better estimate of the solution cannot be reached. This is the recommended option.

Constraint: **Stop Criteria** = SOFT or STRICT.

Stop Tolerance 1 *r* Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$

This option defines ϵ_1 used as a tolerance for the relative duality gap (10) and the relative precision (11), see Section 11.2.

Constraint: **Stop Tolerance 1** $> \epsilon$.

Stop Tolerance 2 *r* Default $= \max\left(10^{-7}, \sqrt{\epsilon}\right)$

This option sets the value ϵ_2 which is used for optimality (12) and complementarity (14) tests from KKT conditions or if **DIMACS Measures** = Check for all DIMACS error measures instead. See Section 11.2.

Constraint: **Stop Tolerance 2** $> \epsilon$.

Stop Tolerance Feasibility *r* Default $= \max\left(10^{-7}, \sqrt{\epsilon}\right)$

This argument places an acceptance limit on the feasibility of the solution (13), ϵ_{feas}. See Section 11.2.

Constraint: **Stop Tolerance Feasibility** $> \epsilon$.

Task *a* Default = MINIMIZE

This argument specifies the required direction of the optimization. If **Task** = FEASIBLE POINT, the objective function (if set) is ignored and the algorithm stops as soon as a feasible point is found with respect to the given tolerance. If no objective function was set, **Task** reverts to FEASIBLE POINT automatically.

Constraint: **Task** = MINIMIZE, MAXIMIZE or FEASIBLE POINT.

Transform Constraints *a* Default = AUTO

This argument controls how equality constraints are treated by the solver. If **Transform Constraints** = EQUALITIES, all equality constraints $h_k(x) = 0$ from (4) are treated as two inequalities $h_k(x) \leq 0$ and $h_k(x) \geq 0$, see Section 11.4. This is the default and the only option in this release for equality constrained problems.

Constraint: **Transform Constraints** = AUTO, NO or EQUALITIES.

U Update Restriction *r* Default = 0.5

This defines the value μ_g giving the bounds on the updates of Lagrangian multipliers for (standard) inequalities between the outer iterations. Values close to 1 limit the changes of the multipliers and serve as a kind of smoothing, lower values allow more significant changes.

Based on numerical experience, big variation in the multipliers may lead to a large number of iterations in the subsequent step and might disturb the convergence due to ill-conditioning.

It might be worth experimenting with the value on your particular problem. Mid range values are recommended over the more extremal ones.

Constraint: $\epsilon <$ **U Update Restriction** < 1.

Umat Update Restriction *r* Default = 0.3

This is an equivalent of **U Update Restriction** for matrix constraints, denoted as μ_A in Section 11.1. The advice above applies equally.

Constraint: $\epsilon <$ **Umat Update Restriction** < 1.