NAG Library Function Document

nag_opt_handle_init (e04rac)

1 Purpose

nag_opt_handle_init (e04rac) initializes a data structure for the NAG optimization modelling suite for problems such as, quadratic programming (QP), nonlinear programming (NLP), 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_init (void **handle, Integer nvar, NagError *fail)
```

3 Description

nag_opt_handle_init (e04rac) initializes an empty problem with n decision variables, x, and returns a handle to the data structure. This handle may then be passed to some of the functions nag_opt_handle_set_linobj (e04rec), nag_opt_handle_set_quadobj (e04rfc), nag_opt_handle_set_nlnobj (e04rgc), nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc), nag_opt_handle_set_nlnconstr (e04rkc), nag_opt_handle_set_nlnhess (e04rlc), nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc) to formulate the problem (define the objective function and constraints) and to a compatible solver, nag_opt_handle_solve_ipopt (e04stc) or nag_opt_handle_solve_pennon (e04svc), to solve it. The handle **must not** be changed between calls. When the handle is no longer needed, nag_opt_handle_free (e04rzc) must be called to destroy it and deallocate all the allocated memory and data within. In addition, the suite comprises auxiliary functions for printing (nag_opt_handle_print (e04ryc)), for setting optional parameters (nag_opt_handle_opt_set (e04zmc) and nag_opt_handle_opt_set_file (e04zpc)), for retrieving them (nag_opt_handle_opt_get (e04znc)) and for reading data files for linear semidefinite programming (nag_opt_sdp_read_sdpa (e04rdc)).

The handle can store various problem formulations, including quadratic programming (QP)

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

nonlinear programming (NLP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) & \text{(a)} \\ \text{subject to} & l_g \leq g(x) \leq u_g & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \tag{2}$$

linear semidefinite programming (SDP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\ldots,m_A & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \tag{3}$$

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

$$
\begin{array}{lll}
\underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\
\text{subject to} & \displaystyle\sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\ldots,m_A & \text{(b)} \\
 & l_B \leq Bx \leq u_B & \text{(c)} \\
 & l_x \leq x \leq u_x, & \text{(d)}
\end{array}
\tag{4}
$$

where H, A_i^k and Q_{ij}^k denote symmetric matrices, B is a general rectangular matrix, m_A is the number of semidefinite constraints (matrix inequalities) and c, l and u are vectors. 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 S should be positive semidefinite.

3.1 Life Cycle of the Handle

Each handle should pass four stages in its life as depicted in the diagram below. These are *initialization*, *problem formulation*, *problem solution* and *deallocation*. The initialization by nag_opt_handle_init (e04rac) and deallocation by nag_opt_handle_free (e04rzc) mark the beginning and the end of the life of the handle. During this time the handle must only be modified by the provided functions. Working with a handle which has not been properly initialized will result in **fail**.**code** = NE_HANDLE (uniform across the suite) and is potentially very dangerous as it may cause unpredictable behaviour.

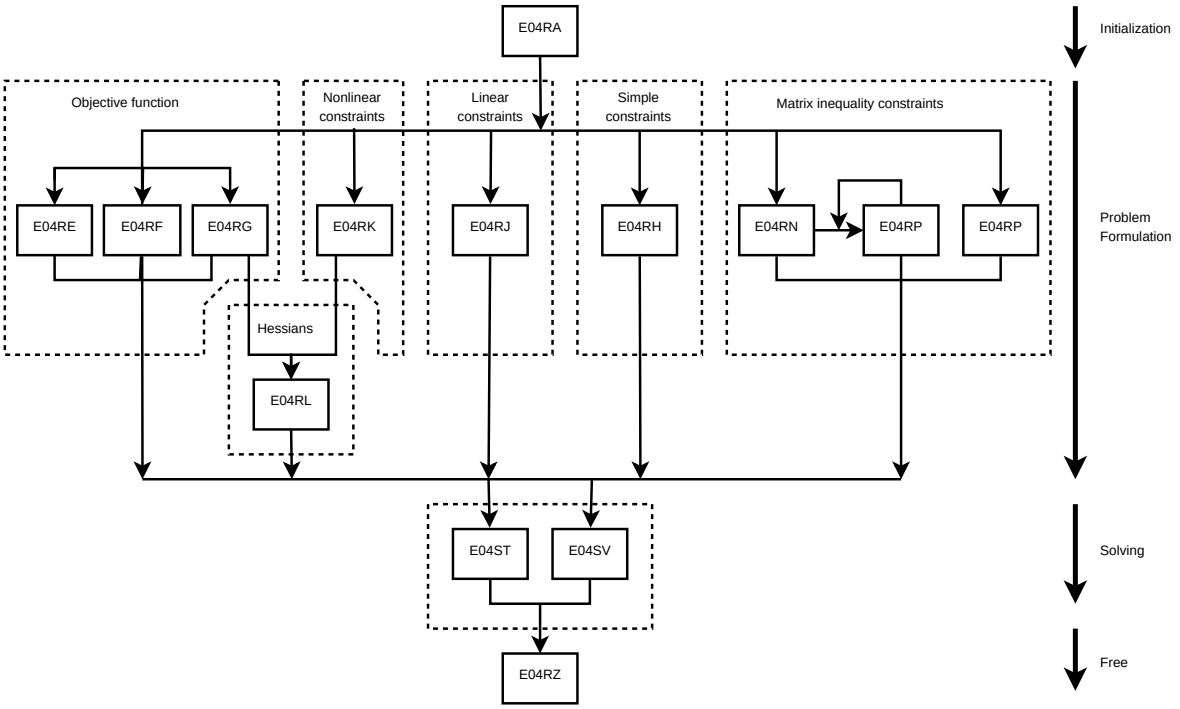
After the handle has been initialized, various routines are provided to add the following basic building blocks to the problem formulation: objective function, simple variable bounds, (standard) linear constraints and matrix constraints. Some of these can be defined at most once (e.g., objective function) and an attempt to redefine them will cause **fail**.**code** = NE_ALREADY_DEFINED. Others (matrix constraints) may be composed by several repetitive calls. The functions work in a tight cooperation, if the provided data is not compatible with the previous information, **fail**.**code** = NE_REF_MATCH is returned.

The handle may be passed to nag_opt_handle_set_linobj (e04rec) to define the linear objective function (3)(a), to nag_opt_handle_set_quadobj (e04rfc) for the quadratic objective function (1)(a), (4)(a), to nag_opt_handle_set_nlnobj (e04rgc) to declare the objective function as a nonlinear function (2)(a) or neither of them if the problem is just to find a feasible point satisfying the constraints. If present, the simple bounds on variables (box constraints, (1)(c), (2)(d), (3)(d), (4)(d)) may be defined by nag_opt_handle_set_simplebounds (e04rhc). The linear constraints ((1)(b), (2)(c), (3)(c) and (4)(c)) are set by nag_opt_handle_set_linconstr (e04rjc). The nonlinear constraints (2)(b) may be declared by nag_opt_handle_set_nlnconstr (e04rkc). If the second derivatives of the nonlinear objective and constraints are available they may be supplied via nag_opt_handle_set_nlnhess (e04rlc). The linear matrix inequalities (3)(b) or the linear part of (4)(b) are defined by nag_opt_handle_set_linmatineq (e04rnc), and this call can be repeated several times if more matrix inequality constraints are required. Any existing (already defined) linear matrix inequalities can be extended by bilinear matrix terms in (4)(b) by one or more calls to nag_opt_handle_set_quadmatineq (e04rpc). The functions nag_opt_handle_set_linobj (e04rec), nag_opt_handle_set_quadobj (e04rfc), nag_opt_handle_set_nlnobj (e04rgc), nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc), nag_opt_handle_set_nlnconstr (e04rkc), nag_opt_handle_set_nlnhess (e04rlc), nag_opt_handle_set_linmatineq (e04rnc) and nag_opt_handle_set_quadmatineq (e04rpc) may be called in an arbitrary order, however, a call to nag_opt_handle_set_linmatineq (e04rnc) must precede a call to nag_opt_handle_set_quadmatineq (e04rpc) for the matrix inequalities with bilinear terms and the nonlinear objective or constraints (nag_opt_handle_set_nlnobj (e04rgc) or nag_opt_handle_set_nlnconstr (e04rkc)) must precede the definition of the second derivatives by nag_opt_handle_set_nlnhess (e04rlc).

When the problem is fully formulated, the handle can be passed to a solver which is compatible with the defined problem. At Mark 26 the NAG optimization modelling suite comprises of nag_opt_handle_solve_ipopt (e04stc) and nag_opt_handle_solve_pennon (e04svc). If the solver cannot deal with the given problem, **fail**.**code** = NE_SETUP_ERROR is returned. Once the solver is called, no further modifications of the problem formulation are allowed and calling any of the functions defining the objective function or the constraints will result in **fail**.**code** = NE_PHASE. The solver may be called repetitively, for example, with various optional parameters and/or starting points.

Any optional parameters may be set by a call to nag_opt_handle_opt_set (e04zmc) at any time between the initialization by nag_opt_handle_init (e04rac) and the call to the solver or after the solver returns. Several optional parameters can be modified at once by nag_opt_handle_opt_set_file (e04zpc) when an option file is used. The current value of the optional parameters may be retrieved by nag_opt_handle_opt_get (e04znc).

For further details, see the documentation of the individual functions and the solvers which also contain a description of all the optional parameters.



4 References

None.

5 Arguments

1: **handle** – void ** *Output*

Note: **handle** does not need to be set on input.

On exit: holds a handle to the internal data structure where an empty problem with **nvar** variables is defined. You **must not** change the handle until the call to nag_opt_handle_free (e04rzc) (deallocation).

2: **nvar** – Integer *Input*

On entry: n, the number of decision variables in the problem.

Constraint: **nvar** > 0.

3: **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_BAD_PARAM

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

NE_INT

On entry, **nvar** = ⟨*value*⟩.
Constraint: **nvar** > 0.

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_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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_handle_init (e04rac) is not threaded in any implementation.

9 Further Comments

None.

10 Example

See examples associated with other routines of the suite:

– the example in Section 10 in nag_opt_sdp_read_sdpa (e04rdc) demonstrates how to use the SDPA file reader and how to solve linear semidefinite programming problems, including printing of the matrix Lagrangian multipliers,

– the example in Section 10 in nag_opt_handle_set_quadobj (e04rfc) presents an alternative way to compute the nearest correlation matrix by means of nonlinear semidefinite programming,

– a matrix completion problem (minimization of a rank of a partially unknown matrix) formulated as SDP is demonsrated in Section 10 in nag_opt_handle_set_simplebounds (e04rhc), the example also demonstrates monitoring mode of the solver nag_opt_handle_solve_pennon (e04svc),

– the example in Section 10 in nag_opt_handle_set_linconstr (e04rjc) solves LP/QP problems read in from an MPS file by nag_opt_miqp_mps_read (e04mxc),

– an application for statistics, E optimal design, solved as an SDP problem is shown in Section 10 in nag_opt_handle_set_linmatineq (e04rnc),

– the example in Section 10 in nag_opt_handle_set_quadmatineq (e04rpc) reads BMI-SDP problem from a file which might be modified by users, in this case it solves Static Output Feedback (SOF) problem,

– the example in Section 10 in nag_opt_handle_print (e04ryc) walks through the life cycle of the handle in which a BMI-SDP problem is formulated and solved,

– an example in Section 10 in nag_opt_handle_solve_ipopt (e04stc) is a small test from Hock and Schittkowski set to show how to call the NLP solver,

– the simple example in Section 10 in nag_opt_handle_solve_pennon (e04svc) demonstrates on the LovÄsz ϑ function eigenvalue optimization problem formulated as SDP.