NAG Library Function Document

nag_opt_handle_solve_ipopt (e04stc)

Note: *this function uses* **optional parameters** *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Sections 1 to 10 of this document. If, however, you wish to reset some or all of the settings please refer to Section 11 for a detailed description of the algorithm and to Section 12 for a detailed description of the specification of the optional parameters.*

1 Purpose

nag_opt_handle_solve_ipopt (e04stc), an interior point method optimization solver, based on the IPOPT software package, is a solver for the NAG optimization modelling suite and is suitable for large scale nonlinear programming (NLP) problems.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_solve_ipopt (void *handle,
     void (*objfun)(Integer nvar, const double x[], double *fx,
          Integer *inform, Nag_Comm *comm),
     void (*objgrd)(Integer nvar, const double x[], Integer nnzfd,
          double fdx[], Integer *inform, Nag_Comm *comm),
     void (*confun)(Integer nvar, const double x[], Integer ncnln,
          double gx[], Integer *inform, Nag_Comm *comm),
     void (*congrd)(Integer nvar, const double x[], Integer nnzgd,
          double gdx[], Integer *inform, Nag_Comm *comm),
     void (*hess)(Integer nvar, const double x[], Integer ncnln, Integer idf,
          double sigma, const double lambda[], Integer nnzh, double hx[],
          Integer *inform, Nag_Comm *comm),
     void (*mon)(Integer nvar, const double x[], Integer nnzu,
          const double u[], Integer *inform, const double rinfo[],
          const double stats[], Nag_Comm *comm),
     Integer nvar, double x[], Integer nnzu, double u[], double rinfo[],
     double stats[], Nag_Comm *comm, NagError *fail)
```

3 Description

nag_opt_handle_solve_ipopt (e04stc) will typically be used for nonlinear programming problems (NLP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) & \text{(a)} \\ \text{subject to} & l_g \le g(x) \le u_g & \text{(b)} \\ & l_B \le Bx \le u_B & \text{(c)} \\ & l_x \le x \le u_x & \text{(d)} \end{array} \qquad (1)$$

where

n is the number of the decision variables,

m_g is the number of the nonlinear constraints and $g(x)$, l_g and u_g are m_g-dimensional vectors,

m_B is the number of the linear constraints and B is a m_B by n matrix, l_B and u_B are m_B-dimensional vectors,

there are n box constraints and l_x and u_x are n-dimensional vectors.

The objective $f(x)$ can be specified in a number of ways: nag_opt_handle_set_linobj (e04rec) for a dense linear function, nag_opt_handle_set_quadobj (e04rfc) for a sparse linear or quadratic function and nag_opt_handle_set_nlnobj (e04rgc) for a general nonlinear function. In the last case, **objfun** and **objgrd** will be used to compute values and gradients of the objective function. Variable box bounds l_x, u_x can be specified with nag_opt_handle_set_simplebounds (e04rhc). The special case of linear constraints l_B, B, u_B is handled by nag_opt_handle_set_linconstr (e04rjc) while general nonlinear constraints $l_g, g(x), u_g$ are specified by nag_opt_handle_set_nlnconstr (e04rkc) (both can be specified). Again, in the last case, **confun** and **congrd** will be used to compute values and gradients of the nonlinear constraint functions.

Finally, if the user is willing to calculate second derivatives, the sparsity structure of the second partial derivatives of a nonlinear objective and/or of any nonlinear constraints is specified by nag_opt_handle_set_nlnhess (e04rlc) while the values of these derivatives themselves will be computed by user-supplied **hess**. While there is an option (see **Hessian Mode**) that forces internal approximation of second derivatives, no such option exists for first derivatives which must be computed accurately. If nag_opt_handle_set_nlnhess (e04rlc) has been called and **hess** is used to calculate values for second derivatives, both the objective and all the constraints must be included; it is not possible to provide a subset of these. If nag_opt_handle_set_nlnhess (e04rlc) is not called, then internal approximation of second derivatives will take place.

3.1 Structure of the Lagrange Multipliers

For a problem consisting of n variable bounds, m_B linear constraints and m_g nonlinear constraints (as specified in **nvar**, **nclin** and **ncnln** of nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc) and nag_opt_handle_set_nlnconstr (e04rkc), respectively), the number of Lagrange multipliers, and consequently the correct value for **nnzu**, will be $q = 2*n + 2*m_B + 2*m_g$. The order these will be found in the **u** array is

$$z_{1_L}, z_{1_U}, z_{2_L}, z_{2_U} \dots z_{n_L}, z_{n_U}, \lambda_{1_L}, \lambda_{1_U}, \lambda_{2_L}, \lambda_{2_U} \dots \lambda_{m_{BL}}, \lambda_{m_{BU}}, \lambda_{(m_B+1)_L}, \lambda_{(m_B+1)_U}, \lambda_{(m_B+2)_L}, \lambda_{(m_B+2)_U} \dots \lambda_{\left(m_B+m_g\right)_L}, \lambda_{\left(m_B+m_g\right)_U}$$

where the L and U subscripts refer to lower and upper bounds, respectively, and the variable bound constraint multipliers come first (if present, i.e., if nag_opt_handle_set_simplebounds (e04rhc) was called), followed by the linear constraint multipliers (if present, i.e., if nag_opt_handle_set_linconstr (e04rjc) was called) and the nonlinear constraint multipliers (if present, i.e., if nag_opt_handle_set_nlnconstr (e04rkc) was called).

Significantly nonzero values for any of these, after the solver has terminated, indicates that the corresponding constraint is active. Significance is judged in the first instance by the relative scale of any value compared to the smallest among them.

4 References

Byrd R H, Gilbert J Ch and Nocedal J (2000) A trust region method based on interior point techniques for nonlinear programming *Mathematical Programming* **89** 149–185

Byrd R H, Liu G and Nocedal J (1997) On the local behavior of an interior point method for nonlinear programming *Numerical Analysis* (eds D F Griffiths and D J Higham) Addison–Wesley

Conn A R, Gould N I M, Orban D and Toint Ph L (2000) A primal-dual trust-region algorithm for non-convex nonlinear programming *Mathematical Programming* **87 (2)** 215–249

Conn A R, Gould N I M and Toint Ph L (2000) *Trust Region Methods* SIAM, Philadephia

Fiacco A V and McCormick G P (1990) *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* SIAM, Philadelphia

Gould N I M, Orban D, Sartenaer A and Toint Ph L (2001) Superlinear convergence of primal-dual interior point algorithms for nonlinear programming *SIAM Journal on Optimization* **11 (4)** 974–1002

Hock W and Schittkowski K (1981) *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer–Verlag

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*

Wl̀chter A and Biegler L T (2006) On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming *Mathematical Programming* **106(1)** 25–57

Williams P and Lang B (2013) A framework for the MR^3 algorithm: theory and implementation *SIAM J. Sci. Comput.* **35** 740–766

Yamashita H (1998) A globally convergent primal-dual interior-point method for constrained optimization *Optimization Methods and Software* **10** 443–469

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_nlnobj (e04rgc), nag_opt_handle_set_simplebounds (e04rhc), nag_opt_handle_set_linconstr (e04rjc), nag_opt_handle_set_nlnconstr (e04rkc) and nag_opt_handle_set_nlnhess (e04rlc). It **must not** be changed between calls to the NAG optimization modelling suite.

2: **objfun** – function, supplied by the user *External Function*

objfun must calculate the value of the nonlinear objective function $f(x)$ at a specified value of the n-element vector of x variables. If there is no nonlinear objective (e.g., nag_opt_handle_set_linobj (e04rec) or nag_opt_handle_set_quadobj (e04rfc) was called to define a linear or quadratic objective function), **objfun** will never be called by nag_opt_handle_solve_ipopt (e04stc) and may be **NULLFN**.

The specification of **objfun** is:

```
void objfun (Integer nvar, const double x[], double *fx,
     Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: the vector x of variable values at which the objective function is to be evaluated.

3: **fx** – double * *Output*

On exit: the value of the objective function at x.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **objfun**. Specifically, if the value is negative then the value of **fx** will be discarded and the solver will either attempt to find a different trial point or terminate immediately with **fail.code** = NE_USER_NAN (the same will happen if **fx** is Infinity or NaN); otherwise, the solver will proceed normally.

5: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **objfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **objfun** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

3: **objgrd** – function, supplied by the user *External Function*

objgrd must calculate the values of the nonlinear objective function gradients $\frac{\partial f}{\partial x}$ at a specified value of the n-element vector of x variables. If there is no nonlinear objective (e.g., nag_opt_handle_set_linobj (e04rec) or nag_opt_handle_set_quadobj (e04rfc) was called to define a linear or quadratic objective function), **objgrd** will never be called by nag_opt_handle_solve_ipopt (e04stc) and may be **NULLFN**.

The specification of **objgrd** is:

```
void objgrd (Integer nvar, const double x[], Integer nnzfd,
     double fdx[], Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: the vector x of variable values at which the objective function gradient is to be evaluated.

3: **nnzfd** – Integer *Input*

On entry: the number of nonzero elements in the sparse gradient vector of the objective function, as was set in a previous call to nag_opt_handle_set_nlnobj (e04rgc).

4: **fdx**[**nnzfd**] – double *Output*

On exit: the values of the nonzero elements in the sparse gradient vector of the objective function, in the order specified by **idxfd** in a previous call to nag_opt_handle_set_nlnobj (e04rgc). **fdx**$[i-1]$ will be the gradient $\frac{\partial f}{\partial x_{\mathbf{idxfd}[i-1]}}$.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **objgrd**. Specifically, if the value is negative the solution of the current problem will terminate immediately with **fail.code** = NE_USER_NAN (the same will happen if **fdx** contains Infinity or NaN); otherwise, computations will continue.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **objgrd**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **objgrd** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

4: **confun** – function, supplied by the user *External Function*

confun must calculate the values of the m_g-element vector $g_i(x)$ of nonlinear constraint functions at a specified value of the n-element vector of x variables. If no nonlinear constraints were registered in this **handle**, **confun** will never be called by nag_opt_handle_solve_ipopt (e04stc) and may be specified as **NULLFN**.

The specification of **confun** is:

```
void confun (Integer nvar, const double x[], Integer ncnln,
     double gx[], Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: the vector x of variable values at which the constraint functions are to be evaluated.

3: **ncnln** – Integer *Input*

On entry: m_g, the number of nonlinear constraints, as specified in an earlier call to nag_opt_handle_set_nlnconstr (e04rkc).

4: **gx**[**ncnln**] – double *Output*

On exit: the m_g values of the nonlinear constraint functions at x.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **confun**. Specifically, if the value is negative then the value of **gx** will be discarded and the solver will either attempt to find a different trial point or terminate immediately with **fail.code** = NE_USER_NAN (the same will happen if **gx** contains Infinity or NaN); otherwise, the solver will proceed normally.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **confun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **confun** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

5: **congrd** – function, supplied by the user *External Function*

congrd must calculate the nonzero values of the sparse Jacobian of the nonlinear constraint functions $\frac{\partial g_i}{\partial x}$ at a specified value of the n-element vector of x variables. If there are no nonlinear constraints (e.g., nag_opt_handle_set_nlnconstr (e04rkc) was never called with the same **handle** or it was called with **ncnln** $= 0$), **congrd** will never be called by nag_opt_handle_solve_ipopt (e04stc) and may be specified as **NULLFN**.

The specification of **congrd** is:

```
void congrd (Integer nvar, const double x[], Integer nnzgd,
     double gdx[], Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: the vector x of variable values at which the Jacobian of the constraint functions is to be evaluated.

3: **nnzgd** – Integer *Input*

On entry: is the number of nonzero elements in the sparse Jacobian of the constraint functions, as was set in a previous call to nag_opt_handle_set_nlnconstr (e04rkc).

4: **gdx**[**nnzgd**] – double *Output*

On exit: the nonzero values of the Jacobian of the nonlinear constraints, in the order specified by **irowgd** and **icolgd** in an earlier call to nag_opt_handle_set_nlnconstr (e04rkc). **gdx**$[i-1]$ will be the gradient $\frac{\partial g_{\mathbf{irowgd}[i-1]}}{\partial x_{\mathbf{icolgd}[i-1]}}$.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **congrd**. Specifically, if the value is negative the solution of the current problem will terminate immediately with **fail.code** = NE_USER_NAN (the same will happen if **gdx** contains Infinity or NaN); otherwise, computations will continue.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **congrd**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **congrd** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

6: **hess** – function, supplied by the user *External Function*

hess must calculate the nonzero values of one of a set of second derivative quantities:

the Hessian of the Lagrangian function $\sigma\nabla^2 f + \sum_{i=1}^{m_g} \lambda_i \nabla^2 g_i$

the Hessian of the objective function $\nabla^2 f$

the Hessian of the constraint functions $\nabla^2 g_i$

The value of argument **idf** determines which one of these is to be computed and this, in turn, is determined by earlier calls to nag_opt_handle_set_nlnhess (e04rlc), when the nonzero sparsity structure of these Hessians was registered. Please note that it is not possible to only supply a subset of the Hessians (see **fail.code** = NE_DERIV_ERRORS or NE_NULL_ARGUMENT). If there were no calls to nag_opt_handle_set_nlnhess (e04rlc), **hess** will never be called by nag_opt_handle_solve_ipopt (e04stc) and may be specified as **NULLFN**. In this case, the Hessian of the Lagrangian will be approximated by a limited-memory quasi-Newton method (L-BFGS).

The specification of **hess** is:

```
void hess (Integer nvar, const double x[], Integer ncnln, Integer idf,
     double sigma, const double lambda[], Integer nnzh, double hx[],
     Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: the vector x of variable values at which the Hessian functions are to be evaluated.

3: **ncnln** – Integer *Input*

On entry: m_g, the number of nonlinear constraints, as specified in an earlier call to nag_opt_handle_set_nlnconstr (e04rkc).

4: **idf** – Integer *Input*

On entry: specifies the quantities to be computed in **hx**.

idf $= -1$
The values of the Hessian of the Lagrangian will be computed in **hx**. This will be the case if nag_opt_handle_set_nlnhess (e04rlc) has been called with **idf** of the same value.

idf $= 0$
The values of the Hessian of the objective function will be computed in **hx**. This will be the case if nag_opt_handle_set_nlnhess (e04rlc) has been called with **idf** of the same value.

idf > 0
The values of the Hessian of the **idf**th constraint function will be computed in **hx**. This will be the case if nag_opt_handle_set_nlnhess (e04rlc) has been called with **idf** of the same value.

5: **sigma** – double *Input*

On entry: if **idf** $= -1$, the value of the σ quantity in the definition of the Hessian of the Lagrangian. Otherwise, **sigma** should not be referenced.

6: **lambda**[**ncnln**] – const double *Input*

On entry: if **idf** $= -1$, the values of the λ_i quantities in the definition of the Hessian of the Lagrangian. Otherwise, **lambda** should not be referenced.

7: **nnzh** – Integer *Input*

On entry: the number of nonzero elements in the Hessian to be computed.

8: **hx**[**nnzh**] – double *Output*

On exit: the nonzero values of the requested Hessian evaluated at x. For each value of **idf**, the ordering of nonzeros must follow the sparsity structure registered in the **handle** by earlier calls to nag_opt_handle_set_nlnhess (e04rlc) through the arguments **irowh** and **icolh**.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **hess**. Specifically, if the value is negative the solution of the current problem will terminate immediately with **fail**.**code** = NE_USER_NAN (the same will happen if **hx** contains Infinity or NaN); otherwise, computations will continue.

10: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **hess**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **hess** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7: **mon** – function, supplied by the user *External Function*

mon is provided to enable you to monitor the progress of the optimization. A facility is provided to halt the optimization process if necessary, using parameter **inform**.

mon may be specified as **NULLFN**.

The specification of **mon** is:

```
void mon (Integer nvar, const double x[], Integer nnzu,
     const double u[], Integer *inform, const double rinfo[],
     const double stats[], Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

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

2: **x**[**nvar**] – const double *Input*

On entry: x^i, the values of the decision variables x at the current iteration.

3: **nnzu** – Integer *Input*

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

4: **u**[**nnzu**] – const double *Input*

On entry: if **nnzu** > 0, **u** holds the values at the current iteration of Lagrange multipliers (dual variables) for the constraints. See Section 3.1 for layout information.

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

On entry: a non-negative value.

On exit: must be set to a value describing the action to be taken by the solver on return from **mon**. Specifically, if the value is negative the solution of the current problem will terminate immediately with **fail**.**code** = NE_USER_STOP; otherwise, computations will continue.

6: **rinfo**[**32**] – const double *Input*

On entry: error measures and various indicators at the end of the current iteration as described in Section 9.1.

7: **stats**[**32**] – const double *Input*

On entry: solver statistics at the end of the current iteration as described in Section 9.1.

8: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **mon**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_handle_solve_ipopt (e04stc) you may allocate memory and initialize these pointers with various quantities for use by **mon** when called from nag_opt_handle_solve_ipopt (e04stc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

8: **nvar** – Integer *Input*

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

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

On entry: x^0, the initial estimates of the variables x.

On exit: the final values of the variables x.

10: **nnzu** – Integer *Input*

On entry: the number of Lagrange multipliers that are to be returned in array **u**.

If **nnzu** $= 0$, **u** will not be referenced; otherwise it needs to match the dimension q as explained in Section 3.1.

Constraints:

nnzu ≥ 0;
if **nnzu** > 0, **nnzu** $= q$.

11: **u**[**nnzu**] – double *Output*

Note: if **nnzu** > 0, **u** holds Lagrange multipliers (dual variables) for the constraints. See Section 3.1 for layout information. If **nnzu** $= 0$, **u** will not be referenced and may be **NULL**.

On exit: the final value of Lagrange multipliers z, λ.

12: **rinfo**[**32**] – double *Output*

On exit: error measures and various indicators at the end of the final iteration as given in the table below:

0	objective function value $f(x)$
1	constraint violation (primal infeasibility) (8)
2	dual infeasibility (7)
3	complementarity
4	Karush–Kuhn–Tucker error

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

On exit: solver statistics at the end of the final iteration as given in the table below:

0	number of the iterations
2	number of backtracking trial steps
3	number of Hessian evaluations
4	number of objective gradient evaluations
7	total wall clock time elapsed
18	number of objective function evaluations
19	number of constraint function evaluations
20	number of constraint Jacobian evaluations

14: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

15: **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 2.3.1.2 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. Initialize a new **handle** using nag_opt_handle_init (e04rac).

NE_BAD_PARAM

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

NE_DERIV_ERRORS

Either all of the constraint and objective Hessian structures must be defined or none (in which case, the Hessians will be approximated by a limited-memory quasi-Newton L-BFGS method).

On entry, a nonlinear objective function has been defined but no objective Hessian sparsity structure has been defined through nag_opt_handle_set_nlnhess (e04rlc).

On entry, a nonlinear constraint function has been defined but no constraint Hessian sparsity structure has been defined through nag_opt_handle_set_nlnhess (e04rlc), for constraint number $\langle value\rangle$.

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_INT

On entry, **nnzu** $= \langle value\rangle$.
Constraint: **nnzu** $= \langle value\rangle$ or 0.

On entry, **nnzu** $= \langle value\rangle$.
Constraint: no constraints present, so **nnzu** must be 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_MAYBE_INFEASIBLE

The solver detected an infeasible problem. *The restoration phase converged to a point that is a minimizer for the constraint violation (in the ℓ_1-norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.*

NE_MAYBE_UNBOUNDED

The solver terminated due to diverging iterates. *The max-norm of the iterates has become larger than a preset value. This can happen if the problem is unbounded below and the iterates are diverging.*

NE_NO_IMPROVEMENT

The solver terminated after the search direction became too small. *This indicates that the solver is calculating very small step sizes and is making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current NLP scaling.*

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_IMPLEMENTED

This routine is not available in this implementation.

NE_NULL_ARGUMENT

The problem requires the **confun** values. Please provide a proper **confun** function.

The problem requires the **congrd** derivatives. Please provide a proper **congrd** function.

The problem requires the **hess** derivatives. Either change the option **Hessian Mode** or provide a proper **hess** function.

The problem requires the **objfun** values. Please provide a proper **objfun** function.

The problem requires the **objgrd** derivatives. Please provide a proper **objgrd** function.

NE_PHASE

The problem is already being solved.

NE_REF_MATCH

The information supplied does not match with that previously stored.
On entry, **nvar** = ⟨*value*⟩ must match that given during initialization of the **handle**, i.e., ⟨*value*⟩.

NE_SETUP_ERROR

This solver does not support matrix inequality constraints.

NE_SUBPROBLEM

The solver terminated after an error in the step computation. *This message is printed if the solver is unable to compute a search direction, despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large. One situation where this can happen is when values in the Hessian are invalid (NaN or Infinity). You can check whether this is true by using the* **Verify Derivatives** *option.*

The solver terminated after failure in the restoration phase. *This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate, does not satisfy the constraint qualification, or if your NLP code provides incorrect derivative information.*

The solver terminated after the maximum time allowed was exceeded. *Maximum number of seconds exceeded. Use option* **Time Limit** *to reset the limit.*

The solver terminated due to an invalid option. Please contact NAG with details of the call to nag_opt_handle_solve_ipopt (e04stc).

The solver terminated due to an invalid problem definition. Please contact NAG with details of the call to nag_opt_handle_solve_ipopt (e04stc).

The solver terminated with not enough degrees of freedom. *This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables.*

NE_TOO_MANY_ITER

Maximum number of iterations exceeded.

NE_USER_NAN

Invalid number detected in user function. *Either **inform** was set to a negative value within the user-supplied functions **objfun**, **objgrd**, **confun**, **congrd** or **hess**, or an Infinity or NaN was detected in values returned from them.*

NE_USER_STOP

User requested termination during a monitoring step. **inform** was set to a negative value in **mon**.

NW_NOT_CONVERGED

The solver reports NLP solved to acceptable level. *This indicates that the algorithm did not converge to the desired tolerances, but that it was able to obtain a point satisfying the acceptable tolerance level. This may happen if the desired tolerances are too small for the current problem.*

7 Accuracy

The accuracy of the solution is driven by optional parameter **Stop Tolerance 1**.

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 $\sqrt{\epsilon}$ where ϵ is the ***machine precision***) 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 11.1 and the description of the particular options.

8 Parallelism and Performance

nag_opt_handle_solve_ipopt (e04stc) is not threaded in any implementation.

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 sent to two independent streams (files) which are set by optional parameters **Print File** and **Monitoring File**. Optional parameters **Print Level** and **Monitoring Level** determine the exposed level of detail. This allows, for example, the generation of a detailed log in a file while the condensed information is displayed on the screen. This section also describes what kind of information is made available to the monitoring function **mon** via **rinfo** and **stats**.

There are four sections printed to the primary output with the default settings (level 2): a derivative check, a header, an iteration log and a summary. At higher levels more information will be printed, including any internal IPOPT options that have been changed from their default values.

Derivative Check

If **Verify Derivatives** is set, then information will appear about any errors detected in the user-supplied derivative functions **objgrd**, **congrd** or **hess**. It may look like this:

```
Starting derivative checker for first derivatives.

 * grad_f[          1] = -2.000000e+00    ˜  2.455000e+01  [ 1.081e+00]
 * jac_g [    1,    4] =  4.700969e+01 v  ˜  5.200968e+01  [ 9.614e-02]
 Starting derivative checker for second derivatives.

 *            obj_hess[    1,    1] =  1.881000e+03 v  ˜  1.882000e+03  [
5.314e-04]
 *     1-th constr_hess[    1,    3] =  2.988964e+00 v  ˜ -1.103543e-02  [
3.000e+00]

 Derivative checker detected 3 error(s).
```

The first line indicates that the value for the partial derivative of the objective with respect to the first variable as returned by **objgrd** (the first one printed) differs sufficiently from a finite difference estimation derived from **objfun** (the second one printed). The number in square brackets is the relative difference between these two numbers.

The second line reports on a discrepancy for the partial derivative of the first constraint with respect to the fourth variable. If the indicator v is absent, the discrepancy refers to a component that had not been included in the sparsity structure, in which case the nonzero structure of the derivatives should be corrected. Mistakes in the first derivatives should be corrected before attempting to correct mistakes in the second derivatives.

The third line reports on a discrepancy in a second derivative of the objective function, differentiated with respect to the first variable, twice.

The fourth line reports on a discrepancy in a second derivative of the first constraint, differentiated with respect to the first and third variables.

Header

If **Print Level** ≥ 1, the header will contain statistics about the size of the problem how the solver sees it, i.e., it reflects any changes imposed by preprocessing and problem transformations. The header may look like:

```
Number of nonzeros in equality constraint Jacobian...:       4Number of
nonzeros in inequality constraint Jacobian.:       8Number of nonzeros in
Lagrangian Hessian.............:      10Total number of vari-
ables............................:       4                  variables with
only lower bounds:       4             variables with lower and upper
bounds:       0                    variables with only upper bounds:
0Total number of equality constraints.................:       1Total number
of inequality constraints...............:       2       inequality
constraints with only lower bounds:       2   inequality constraints with
lower and upper bounds:       0      inequality constraints with only upper
bounds:       0
```

It summarises what is known about the variables and the constraints. Simple bounds are set by nag_opt_handle_set_simplebounds (e04rhc) and standard equalities and inequalities by nag_opt_han dle_set_linconstr (e04rjc).

Iteration log

If **Print Level** $= 2$, the status of each iteration is condensed to one line. The line shows:

`iter`	The current iteration count. This includes regular iterations and iterations during the restoration phase. If the algorithm is in the restoration phase, the letter **r** will be appended to the iteration number. The iteration number 0 represents the starting point. This quantity is also available as **stats**[0] of **mon**.
`objective`	The unscaled objective value at the current point (given the current NLP scaling). During the restoration phase, this value remains the unscaled objective value for the original problem. This quantity is also available as **rinfo**[0] of **mon**.
`inf_pr`	The unscaled constraint violation at the current point (given the current NLP scaling). This quantity is the infinity-norm (max) of the (unscaled) constraints g_i. During the restoration phase, this value remains the constraint violation of the original problem at the current point. This quantity is also available as **rinfo**[1] of **mon**.
`inf_du`	The scaled dual infeasibility at the current point (given the current NLP scaling). This quantity measure the infinity-norm (max) of the internal dual infeasibility, λ_i of Eq. (4a) in the implementation paper WÌchter and Biegler (2006), including inequality constraints reformulated using slack variables and NLP scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem. This quantity is also available as **rinfo**[2] of **mon**.
`lg(mu)`	$log10$ of the value of the barrier parameter μ. μ itself is also available as **rinfo**[3] of **mon**.
`||d||`	The infinity norm (max) of the primal step (for the original variables x and the internal slack variables s). During the restoration phase, this value includes the values of additional variables, $\bar{p}$ and $\bar{n}$ (see Eq. (30) in WÌchter and Biegler (2006)). This quantity is also available as **rinfo**[4] of **mon**.
`lg(rg)`	$log10$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (δ_w of Eq. (26) and Section 3.1 in WÌchter and Biegler (2006)). A dash (–) indicates that no regularization was done. The regularization term itself is also available as **rinfo**[5] of **mon**.
`alpha_du`	The stepsize for the dual variables (α_k^z of Eq. (14c) in WÌchter and Biegler (2006)). This quantity is also available as **rinfo**[6] of **mon**.
`alpha_pr`	The stepsize for the primal variables (α_k of Eq. (14a) in WÌchter and Biegler (2006)). This quantity is also available as **rinfo**[7] of **mon**. The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion.

ff-type iteration in the filter method without second order correction

Ff-type iteration in the filter method with second order correction

hh-type iteration in the filter method without second order correction

Hh-type iteration in the filter method with second order correction

kpenalty value unchanged in merit function method without second order correction

Kpenalty value unchanged in merit function method with second order correction

npenalty value updated in merit function method without second order correction

Npenalty value updated in merit function method with second order correction

RRestoration phase just started

win watchdog procedure

sstep accepted in soft restoration phase

t/Ttiny step accepted without line search

rsome previous iterate restored

`ls` The number of backtracking line search steps (does not include second order correction steps). This quantity is also available as **stats**[1] of **mon**.

Note that the step acceptance mechanisms in IPOPT consider the barrier objective function (5) which is usually different from the value reported in the `objective` column. Similarly, for the purposes of the step acceptance, the constraint violation is measured for the internal problem formulation, which includes slack variables for inequality constraints and potentially NLP scaling of the constraint functions. This value, too, is usually different from the value reported in `inf_pr`. As a consequence, a new iterate might have worse values both for the objective function and the constraint violation as reported in the iteration output, seemingly contradicting globalization procedure.

Note that all these values are also available in **rinfo**[0], ..., **rinfo**[7] and **stats**[0], ..., **stats**[1]of the monitoring function **mon**.

The output might look as follows:

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
    0  2.6603500e+05 1.55e+02 3.21e+01  -1.0 0.00e+00    -  0.00e+00 0.00e+00
0
    1  1.5053889e+05 7.95e+01 1.43e+01  -1.0 1.16e+00    -  4.55e-01 1.00e+00f
1
    2  8.9745785e+04 3.91e+01 6.45e+00  -1.0 3.07e+01    -  5.78e-03 1.00e+00f
1
    3  3.9878595e+04 1.63e+01 3.47e+00  -1.0 5.19e+00   0.0 2.43e-01 1.00e
+00f  1
    4  2.7780042e+04 1.08e+01 1.64e+00  -1.0 3.66e+01    -  7.24e-01 8.39e-01f
1
    5  2.6194274e+04 1.01e+01 1.49e+00  -1.0 1.07e+01    -  1.00e+00 1.05e-01f
1
    6  1.5422960e+04 4.75e+00 6.82e-01  -1.0 1.74e+01    -  1.00e+00 1.00e+00f
1
    7  1.1975453e+04 3.14e+00 7.26e-01  -1.0 2.83e+01    -  1.00e+00 5.06e-01f
1
    8  8.3508421e+03 1.34e+00 2.04e-01  -1.0 3.96e+01    -  9.27e-01 1.00e+00f
1
    9  7.0657495e+03 4.85e-01 9.22e-02  -1.0 5.32e+01    -  1.00e+00 1.00e+00f
1
 iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
   10  6.8359393e+03 1.17e-01 1.28e-01  -1.7 4.69e+01    -  8.21e-01 1.00e+00h
```

```
1
   11  6.6508917e+03 1.52e-02 1.52e-02  -2.5 1.87e+01    -  1.00e+00 1.00e+00h
1
   12  6.4123213e+03 8.77e-03 1.49e-01  -3.8 1.85e+01    -  7.49e-01 1.00e+00f
1
   13  6.3157361e+03 4.33e-03 1.90e-03  -3.8 2.07e+01    -  1.00e+00 1.00e+00f
1
   14  6.2989280e+03 1.12e-03 4.06e-04  -3.8 1.54e+01    -  1.00e+00 1.00e+00h
1
   15  6.2996264e+03 9.90e-05 2.05e-04  -5.7 5.35e+00    -  9.63e-01 1.00e+00h
1
   16  6.2998436e+03 0.00e+00 1.86e-07  -5.7 4.55e-01    -  1.00e+00 1.00e+00h
1
   17  6.2998424e+03 0.00e+00 6.18e-12  -8.2 2.62e-03    -  1.00e+00 1.00e+00h
1
```

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

Summary

Once the solver finishes, a detailed summary is produced if **Print Level** ≥ 1. An example is shown below:

```
Number of Iterations....: 6

                                   (scaled)                 (unscaled)
 Objective...............:   7.8692659500479623e-01    6.2324586324379867e
+00
 Dual infeasibility......:   7.9744615766675617e-10    6.3157735687207093e-09
 Constraint violation....:   8.3555384833289281e-12    8.3555384833289281e-12
 Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e
+00
 Overall NLP error.......:   7.9744615766675617e-10    6.3157735687207093e-09


 Number of objective function evaluations             = 7
 Number of objective gradient evaluations             = 7
 Number of equality constraint evaluations            = 7
 Number of inequality constraint evaluations          = 0
 Number of equality constraint Jacobian evaluations   = 7
 Number of inequality constraint Jacobian evaluations = 0
 Number of Lagrangian Hessian evaluations             = 6
 Total CPU secs in IPOPT (w/o function evaluations)   =      0.724
 Total CPU secs in NLP function evaluations           =      0.343

 EXIT: Optimal Solution Found.
```

It starts with the total number of iterations the algorithm went through. Then, five quantities are printed, all evaluated at the termination point: the value of the objective function, the dual infeasibility, the constraint violation, the complementarity and the NLP error.

This is followed by some statistics on the number of calls to user-supplied functions and CPU time taken in user-supplied functions and the main algorithm. Lastly, status at exit is indicated by a short message. Detailed timings of the algorithm are displayed only if **Stats Time** is set.

9.2 Additional Licensor

Parts of the code for nag_opt_handle_solve_ipopt (e04stc) are distributed according to terms imposed by another licensor. Please refer to the list of Library licensors available on the NAG Website for further details.

10 Example

This example is based on Problem 73 in Hock and Schittkowski (1981) and involves the minimization of the linear function

$$f(x) = 24.55x_1 + 26.75x_2 + 39.00x_3 + 40.50x_4$$

subject to the bounds

$$\begin{array}{l} 0 \leq x_1, \\ 0 \leq x_2, \\ 0 \leq x_3, \\ 0 \leq x_4, \end{array}$$

to the nonlinear constraint

$$12x_1 + 11.9x_2 + 41.8x_3 + 52.1x_4 - 21 - 1.645\sqrt{0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2} \geq 0$$

and the linear constraints

$$\begin{array}{c} 2.3x_1 + 5.6x_2 + 11.1x_3 + 1.3x_4 \geq 5, \\ x_1 + x_2 + x_3 + x_4 - 1 = 0. \end{array}$$

The initial point, which is infeasible, is

$$x_0 = \begin{pmatrix} 1, & 1, & 1, & 1 \end{pmatrix}^{\mathrm{T}}$$

and $f(x_0) = 130.8$. The optimal solution (to five significant figures) is

$$x^* = (0.63552, 0.0, 0.31270, 0.051777)^{\mathrm{T}},$$

10.1 Program Text

```
/* nag_opt_handle_solve_ipopt (e04stc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/*
 * NLP example: Linear objective + Linear constraint + Non-Linear constraint
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagf16.h>
#include <nagx04.h>
#include <math.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL objfun(Integer nvar, const double x[],
                              double *fx,
                              Integer *flag, Nag_Comm *comm);
  static void NAG_CALL objgrd(Integer nvar, const double x[],
                              Integer nnzfd, double fdx[],
                              Integer *flag, Nag_Comm *comm);
  static void NAG_CALL confun(Integer nvar, const double x[],
                              Integer ncnln, double gx[],
                              Integer *flag, Nag_Comm *comm);
  static void NAG_CALL congrd(Integer nvar, const double x[],
                            Integer nnzgd, double gdx[],
                            Integer *flag, Nag_Comm *comm);
  static void NAG_CALL hess(Integer nvar, const double x[],
                          Integer ncnln, Integer idf, double sigma,
                          const double lambda[], Integer nnzh, double hx[],
                          Integer *flag, Nag_Comm *comm);
  static void NAG_CALL mon(Integer nvar, const double x[],
                         Integer nnzu, const double u[],
                         Integer *flag, const double rinfo[],
                         const double stats[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  #define BIGBND 1.0E40
  /* Scalars */
  double     solve_timeout = 5.0;
  Integer    exit_status = 0, islinear;
  Integer    i, idlc, idx, j, nnzu, nvar, nclin, ncnln, nnzgd;

  /* Arrays */
  Integer    iuser[2];
  Integer    idxfd[4] = {1,2,3,4};
  double     ruser[4] = {24.55, 26.75, 39.00, 40.50};
  double     rinfo[32], stats[32], *x = 0, *u = 0;
  double     bl[4] = {0,0,0,0};
  double     bu[4] = {BIGBND,BIGBND,BIGBND,BIGBND};
  double     linbl[2] = {5.0,1.0};
  double     linbu[2] = {BIGBND,1.0};
  double     nlnbl[1] = {21.0};
  double     nlnbu[1] = {BIGBND};
  Integer    irowb[8] = {1,1,1,1,2,2,2,2};
  Integer    icolb[8] = {1,2,3,4,1,2,3,4};
  Integer    irowgd[4] = {1,1,1,1};
  Integer    icolgd[4] = {1,2,3,4};
  Integer    irowh[10], icolh[10];
  double     b[8]= {2.3, 5.6, 11.1, 1.3, 1.0, 1.0, 1.0, 1.0};
  char       opt[80];
  void       *handle = 0;

  /* Nag Types */
  NagError   fail;
  Nag_Comm   comm;
  Nag_FileID nout, file_out, mon_out, umon_out;

  /* nag_open_file (x04acc).
   *  Open unit number for reading, writing or appending, and
   *  associate unit with named file
   */
  nag_open_file("", 1, &nout, NAGERR_DEFAULT);

  /* nag_write_line (x04bac).
   * Write formatted record to external file
   */
  nag_write_line(nout, "nag_opt_handle_solve_ipopt (e04stc) "
                 "Example Program Results");

  nag_open_file("e04stc.out", 1, &file_out, NAGERR_DEFAULT);
  nag_open_file("e04stc.mon", 1, &mon_out, NAGERR_DEFAULT);
  nag_open_file("e04stc.umon", 1, &umon_out, NAGERR_DEFAULT);

  for (islinear=0;islinear<=1;islinear++){
    nnzu = 0;
    nvar = 4;
    /* nag_opt_handle_init (e04rac).
     * Initialize an empty problem handle with NVAR variables.
     */
    nag_opt_handle_init(&handle, nvar, NAGERR_DEFAULT);
```

```
sprintf(opt,"Infinite Bound Size = %e",BIGBND);
/* nag_opt_handle_opt_set (e04zmc).
 * Set optional arguments of the solver
 */
nag_opt_handle_opt_set(handle, opt, NAGERR_DEFAULT);

nnzu += 2*nvar;
/* nag_opt_handle_set_simplebounds (e04rhc).
 * Define bounds on the variables
 */
nag_opt_handle_set_simplebounds(handle, nvar, bl, bu, NAGERR_DEFAULT);

iuser[0] = islinear;
comm.iuser = iuser;
comm.user = ruser;
if (islinear==1) {
  /* nag_opt_handle_set_linobj (e04rec).
   * Define linear objective
   */
  nag_opt_handle_set_linobj(handle, nvar, ruser, NAGERR_DEFAULT);
} else {
  /* nag_opt_handle_set_nlnobj (e04rgc).
   * Define non-linear objective
   */
  nag_opt_handle_set_nlnobj(handle, nvar, idxfd, NAGERR_DEFAULT);
}
nclin = 2;
nnzu += 2*nclin;
idlc = 0;
/* nag_opt_handle_set_linconstr (e04rjc).
 * Define a block of linear constraints
 */
nag_opt_handle_set_linconstr(handle, nclin, linbl, linbu, nclin*nvar,
                             irowb, icolb, b, &idlc, NAGERR_DEFAULT);

ncnln = 1;
/* dense gradients */
nnzgd = ncnln * nvar;
nnzu += 2*ncnln;
/* nag_opt_handle_set_nlnconstr (e04rkc).
 * Define a block of nonlinear constraints
 */
nag_opt_handle_set_nlnconstr(handle, ncnln, nlnbl, nlnbu, nnzgd,
                             irowgd, icolgd, NAGERR_DEFAULT);

/* Define structure of the Hessian, dense upper triangle */
for(idx=0,i=1;i<=nvar;i++)
  for(j=i;j<=nvar;j++,idx++){
    icolh[idx] = j;
    irowh[idx] = i;
}
/* nag_opt_handle_set_nlnhess (e04rlc).
 * Define structure of Hessian of objective, constraints or the Lagrangian
 */
nag_opt_handle_set_nlnhess(handle, 1, idx, irowh, icolh, NAGERR_DEFAULT);
if (islinear!=1)
  nag_opt_handle_set_nlnhess(handle, 0, idx, irowh, icolh, NAGERR_DEFAULT);
/* nag_opt_handle_print (e04ryc).
   Print information about a problem
 */
nag_opt_handle_print(handle, nout, "Overview", NAGERR_DEFAULT);

if (!(x = NAG_ALLOC(nvar, double)) ||
    !(u = NAG_ALLOC(nnzu, double))) {
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

for (i=0;i<nvar;i++) x[i]=1.0;
iuser[1] = umon_out;
```

```
  sprintf(opt, "Monitoring File = %" NAG_IFMT, mon_out);
  nag_opt_handle_opt_set(handle, opt, NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Monitoring Level = 3", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Outer Iteration Limit = 26",
                         NAGERR_DEFAULT);

  sprintf(opt, "Print File = %" NAG_IFMT, file_out);
  nag_opt_handle_opt_set(handle, opt, NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Print Level = 2", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Stop Tolerance 1 = 2.5e-8", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Time Limit = 60", NAGERR_DEFAULT);

  INIT_FAIL(fail);
  /* nag_opt_handle_solve_ipopt (e04stc).
   * Run Ipopt solver on a sparse nonlinear programming problem
   */
  nag_opt_handle_solve_ipopt(handle, objfun, objgrd, confun, congrd, hess,
                             mon, nvar, x, nnzu, u, rinfo, stats, &comm,
                             &fail);
  if (fail.code == NE_NOERROR) {
    char msg[80];
    nag_opt_handle_print(handle, nout, "Options", &fail);
    nag_write_line(nout, "Variables");
    for (i=0; i<nvar; i++) {
      sprintf(msg, "     x(%10" NAG_IFMT ")%17s=%20.6e", i+1, "", x[i]);
      nag_write_line(nout, msg);
    }
    nag_write_line(nout, "Variable bound Lagrange multipliers");
    for (i=0; i<nvar; i++) {
      sprintf(msg, "     zL(%10" NAG_IFMT ")%16s=%20.6e", i+1, "", u[2*i]);
      nag_write_line(nout, msg);
      sprintf(msg, "     zU(%10" NAG_IFMT ")%16s=%20.6e", i+1, "", u[2*i+1]);
      nag_write_line(nout, msg);
    }
    nag_write_line(nout, "Linear constraints Lagrange multipliers");
    for (i=0;i<nclin;i++) {
      sprintf(msg, "     l+(%10" NAG_IFMT ")%16s=%20.6e", i+1, "",
              u[2*nvar+2*i]);
      nag_write_line(nout,msg);
      sprintf(msg, "     l-(%10" NAG_IFMT ")%16s=%20.6e", i+1, "",
              u[2*nvar+2*i+1]);
      nag_write_line(nout, msg);
    }
    nag_write_line(nout, "Nonlinear constraints Lagrange multipliers");
    for (i=0;i<ncnln;i++) {
      sprintf(msg, "     l+(%10" NAG_IFMT ")%16s=%20.6e", i+1, "",
              u[2*(nvar+nclin)+2*i]);
      nag_write_line(nout, msg);
      sprintf(msg, "     l-(%10" NAG_IFMT ")%16s=%20.6e", i+1, "",
              u[2*(nvar+nclin)+2*i+1]);
      nag_write_line(nout, msg);
    }
    sprintf(msg, "\nAt solution, Objective minimum     =%20.7e", rinfo[0]);
    nag_write_line(nout, msg);
    sprintf(msg, "             Constraint violation  =%20.2e", rinfo[1]);
    nag_write_line(nout, msg);
    sprintf(msg, "             Dual infeasibility    =%20.2e", rinfo[2]);
    nag_write_line(nout, msg);
    sprintf(msg, "             Complementarity       =%20.2e", rinfo[3]);
    nag_write_line(nout, msg);
    sprintf(msg, "             KKT error             =%20.2e", rinfo[4]);
    nag_write_line(nout, msg);
    if (stats[7] < solve_timeout)
      sprintf(msg, "Solved in allotted time limit");
    else
      sprintf(msg, "Solution took %6.2f sec, which is longer than expected",
              stats[7]);
    nag_write_line(nout, msg);
    sprintf(msg, "    after iterations                       :%11.0f",
            stats[0]);
      nag_write_line(nout, msg);
      sprintf(msg, "    after objective evaluations            :%11.0f",
              stats[18]);
      nag_write_line(nout, msg);
      sprintf(msg, "    after objective gradient evaluations   :%11.0f",
              stats[4]);
      nag_write_line(nout, msg);
      sprintf(msg, "    after constraint evaluations           :%11.0f",
              stats[19]);
      nag_write_line(nout, msg);
      sprintf(msg, "    after constraint gradient evaluations  :%11.0f",
              stats[20]);
      nag_write_line(nout, msg);
      sprintf(msg, "    after hessian evaluations              :%11.0f",
              stats[3]);
      nag_write_line(nout, msg);
      nag_opt_handle_print(handle, nout, "Overview", NAGERR_DEFAULT);
      nag_write_line(nout,
                     "------------------------------------------------------");
    } else {
      printf("Error from nag_opt_handle_solve_ipopt (e04stc).\n%s\n",
             fail.message);
      exit_status = 1;
    }
    if (handle)
      /* nag_opt_handle_free (e04rzc).
       * Destroy the problem handle and deallocate all the memory used
       */
      nag_opt_handle_free(&handle, NAGERR_DEFAULT);

    NAG_FREE(x);
    NAG_FREE(u);
  }

 END:
  return exit_status;
}

/* Subroutine */
#include <assert.h>
  static void NAG_CALL objfun(Integer nvar, const double x[],
                              double *fx,
                              Integer *flag, Nag_Comm *comm)
  {
    *flag = 0;
    /* nag_ddot (f16eac).
     * Dot product of two vectors, allows scaling and accumulation
     */
    nag_ddot(Nag_NoConj,4,1.0,x,1,0.0,comm->user,1,fx,NAGERR_DEFAULT);
    assert (comm->iuser[0]!=1) ;
  }
  static void NAG_CALL objgrd(Integer nvar, const double x[],
                              Integer nnzfd, double fdx[],
                              Integer *flag, Nag_Comm *comm)
  {
    *flag = 0;
    /* nag_dge_copy (f16qfc).
     * Matrix copy, real rectangular matrix
     */
    nag_dge_copy(Nag_ColMajor, Nag_NoTrans, nnzfd, 1, comm->user, nnzfd, fdx,
                 nnzfd, NAGERR_DEFAULT);
    assert(comm->iuser[0]!=1);
  }
  static void NAG_CALL confun(Integer nvar, const double x[],
                              Integer ncnln, double gx[],
                              Integer *flag, Nag_Comm *comm)
  {
    *flag = 0;
    gx[0]= 12.0*x[0] + 11.9*x[1] + 41.8*x[2] + 52.1*x[3] -
      1.645*sqrt(.28*x[0]*x[0]+.19*x[1]*x[1]+20.5*x[2]*x[2]+.62*x[3]*x[3]);
  }
```

```
static void NAG_CALL congrd(Integer nvar, const double x[],
                            Integer nnzgd, double gdx[],
                            Integer *flag, Nag_Comm *comm)
{
  double tmp;
  *flag = 0;
  tmp  = sqrt(0.62*x[3]*x[3]+20.5*x[2]*x[2]+0.19*x[1]*x[1]+0.28*x[0]*x[0]);
  gdx[0] = (12.0*tmp-0.4606*x[0])/tmp;
  gdx[1] = (11.9*tmp-0.31255*x[1])/tmp;
  gdx[2] = (41.8*tmp-33.7225*x[2])/tmp;
  gdx[3] = (52.1*tmp-1.0199*x[3])/tmp;
}
static void NAG_CALL hess(Integer nvar, const double x[], Integer ncnln,
                          Integer idf, double sigma, const double lambda[],
                          Integer nnzh, double hx[],
                          Integer *flag, Nag_Comm *comm)
{
  double tmp;
  *flag = 0;
  /* nag_dload (f16fbc).
   * Broadcast scalar into real vector
   */
  nag_dload(nnzh, 0.0, hx, 1, NAGERR_DEFAULT);

  if (idf==0) return; /* objective is linear */
  tmp = sqrt(0.62*x[3]*x[3] + 20.5*x[2]*x[2] + 0.19*x[1]*x[1]
           + 0.28*x[0]*x[0]);
  tmp = tmp*(x[3]*x[3] + 33.064516129032258064 *x[2]*x[2]
                       + 0.306451612903225806 45*x[1]*x[1]
                       + 0.4516129032258064 5161*x[0]*x[0]);
  /* Col 1 */
  hx[0] = (-0.4606*x[3]*x[3] - 15.229516129032258064 *x[2]*x[2]
                             - 0.14115161290322580645*x[1]*x[1])/tmp;
  hx[1] = (0.14115161290322580645*x[0]*x[1])/tmp;
  hx[2] = (15.229516129032258064 *x[0]*x[2])/tmp;
  hx[3] = (0.4606*x[0]*x[3])/tmp;
  /* Col 2 */
  hx[4] = (-0.31255*x[3]*x[3] - 10.334314516129032258 *x[2]*x[2]
                              - 0.14115161290322580645*x[0]*x[0])/tmp;
  hx[5] = (10.334314516129032258*x[1]*x[2])/tmp;
  hx[6] = (0.31255*x[1]*x[3])/tmp;
  /* Col 3 */
  hx[7] = (-33.7225*x[3]*x[3] - 10.334314516129032258*x[1]*x[1]
                              - 15.229516129032258065*x[0]*x[0])/tmp;
  hx[8] = (33.7225*x[2]*x[3])/tmp;
  /* Col 4 */
  hx[9] = (-33.7225*x[2]*x[2] - 0.31255*x[1]*x[1] - 0.4606*x[0]*x[0])/tmp;
  /* nag_daxpby (f16ecc).
   * Real weighted vector addition
   */
  if (idf==-1)
    nag_daxpby(nnzh, 0.0, hx, 1, lambda[0], hx, 1, NAGERR_DEFAULT);
  else
    assert(idf == 1);
}
static void NAG_CALL mon(Integer nvar, const double x[],
                         Integer nnzu, const double u[],
                         Integer *flag, const double rinfo[],
                         const double stats[], Nag_Comm *comm)
{
  Integer i;
  char    msg[80];
  char    fmt[80] = "%2" NAG_IFMT "%14.6e %2" NAG_IFMT "%14.6e ";

  *flag = 0;
  nag_write_line(comm->iuser[1], "Monitoring...");
  nag_write_line(comm->iuser[1], "   x[]");
  for (i=0; i<nvar; i+=2) {
    sprintf(msg, fmt, i, x[i], i+1, x[i+1]);
    nag_write_line(comm->iuser[1], msg);
  }
```

```
    nag_write_line(comm->iuser[1], "   u[]");
    for (i=0; i<nnzu; i+=2) {
      sprintf(msg, fmt, i, u[i], i+1, u[i+1]);
      nag_write_line(comm->iuser[1], msg);
    }
    nag_write_line(comm->iuser[1], "   rinfo[31]");
    for (i=0; i<32; i+=2) {
      sprintf(msg, fmt, i, rinfo[i], i+1, rinfo[i+1]);
      nag_write_line(comm->iuser[1], msg);
    }
    nag_write_line(comm->iuser[1], "   stats[31]");
    for (i=0; i<32; i+=2) {
      sprintf(msg, fmt, i, stats[i], i+1, stats[i+1]);
      nag_write_line(comm->iuser[1], msg);
    }
  }
```

10.2 Program Results

```
nag_opt_handle_solve_ipopt (e04stc) Example Program Results
 Overview
   Status:                  Problem and option settings are editable.
   No of variables:         4
   Objective function:      nonlinear
   Simple bounds:           defined
   Linear constraints:      2
   Nonlinear constraints:   1
   Matrix constraints:      not defined yet
 Option settings
 Begin of Options
     Outer Iteration Limit         =                  26     * U
     Infinite Bound Size           =         1.00000E+40     * U
     Print File                    =                  10     * U
     Print Level                   =                   2     * U
     Monitoring File               =                  11     * U
     Monitoring Level              =                   3     * U
     Stats Time                    =                  No     * d
     Stop Tolerance 1              =         2.50000E-08     * U
     Hessian Mode                  =               Exact     * S
     Verify Derivatives            =                 Yes     * S
     Time Limit                    =         6.00000E+01     * U
 End of Options
Variables
     x(         1)                 =        6.355216e-01
     x(         2)                 =        2.066279e-10
     x(         3)                 =        3.127019e-01
     x(         4)                 =        5.177655e-02
Variable bound Lagrange multipliers
     zL(         1)                =        3.916168e-09
     zU(         1)                =        0.000000e+00
     zL(         2)                =        2.433326e-01
     zU(         2)                =        0.000000e+00
     zL(         3)                =        7.974843e-09
     zU(         3)                =        0.000000e+00
     zL(         4)                =        4.944607e-08
     zU(         4)                =        0.000000e+00
Linear constraints Lagrange multipliers
     l+(         1)                =        0.000000e+00
     l-(         1)                =        4.105411e-01
     l+(         2)                =        0.000000e+00
     l-(         2)                =        5.803551e-01
Nonlinear constraints Lagrange multipliers
     l+(         1)                =        0.000000e+00
     l-(         1)                =        1.837124e+01

At solution, Objective minimum     =       2.9894378e+01
             Constraint violation  =            1.11e-16
             Dual infeasibility    =            6.72e-12
             Complementarity       =            2.56e-09
             KKT error             =            2.56e-09
```

```
Solved in allotted time limit
    after iterations                        :          8
    after objective evaluations             :          9
    after objective gradient evaluations    :          9
    after constraint evaluations            :          9
    after constraint gradient evaluations   :          9
    after hessian evaluations               :          8
 Overview
   Status:                 Solver finished, only options can be changed.
   No of variables:        4
   Objective function:     nonlinear
   Simple bounds:          defined
   Linear constraints:     2
   Nonlinear constraints:  1
   Matrix constraints:     not defined
-----------------------------------------------------
 Overview
   Status:                 Problem and option settings are editable.
   No of variables:        4
   Objective function:     linear
   Simple bounds:          defined
   Linear constraints:     2
   Nonlinear constraints:  1
   Matrix constraints:     not defined yet
 Option settings
 Begin of Options
     Outer Iteration Limit         =                  26     * U
     Infinite Bound Size           =         1.00000E+40     * U
     Print File                    =                  10     * U
     Print Level                   =                   2     * U
     Monitoring File               =                  11     * U
     Monitoring Level              =                   3     * U
     Stats Time                    =                  No     * d
     Stop Tolerance 1              =         2.50000E-08     * U
     Hessian Mode                  =               Exact     * S
     Verify Derivatives            =                 Yes     * S
     Time Limit                    =         6.00000E+01     * U
 End of Options
Variables
     x(         1)                 =        6.355216e-01
     x(         2)                 =        2.066279e-10
     x(         3)                 =        3.127019e-01
     x(         4)                 =        5.177655e-02
Variable bound Lagrange multipliers
     zL(         1)                =        3.916168e-09
     zU(         1)                =        0.000000e+00
     zL(         2)                =        2.433326e-01
     zU(         2)                =        0.000000e+00
     zL(         3)                =        7.974843e-09
     zU(         3)                =        0.000000e+00
     zL(         4)                =        4.944607e-08
     zU(         4)                =        0.000000e+00
Linear constraints Lagrange multipliers
     l+(         1)                =        0.000000e+00
     l-(         1)                =        4.105411e-01
     l+(         2)                =        0.000000e+00
     l-(         2)                =        5.803551e-01
Nonlinear constraints Lagrange multipliers
     l+(         1)                =        0.000000e+00
     l-(         1)                =        1.837124e+01

At solution, Objective minimum     =       2.9894378e+01
             Constraint violation  =            1.11e-16
             Dual infeasibility    =            6.72e-12
             Complementarity       =            2.56e-09
             KKT error             =            2.56e-09
Solved in allotted time limit
    after iterations                        :          8
    after objective evaluations             :          9
    after objective gradient evaluations    :          9
    after constraint evaluations            :          9
    after constraint gradient evaluations   :         9
    after hessian evaluations               :         8
 Overview
   Status:                 Solver finished, only options can be changed.
   No of variables:        4
   Objective function:     linear
   Simple bounds:          defined
   Linear constraints:     2
   Nonlinear constraints:  1
   Matrix constraints:     not defined
----------------------------------------------------
```

11 Algorithmic Details

nag_opt_handle_solve_ipopt (e04stc) is an implementation of IPOPT (see WÌchter and Biegler (2006)) that is fully supported and maintained by NAG. It uses Harwell packages MA97 for the underlying sparse linear algebra factorization and MC68 approximate minimum degree algorithm for the ordering. Any issues relating to nag_opt_handle_solve_ipopt (e04stc) should be directed to NAG who assume all responsibility for the nag_opt_handle_solve_ipopt (e04stc) function and its implementation.

In the remainder of this section, we repeat part of Section 2.1 of WÌchter and Biegler (2006).

To simplify notation, we describe the method for the problem formulation

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{2}$$

$$\text{subject to} \quad g(x) = 0 \tag{3}$$

$$x \geq 0. \tag{4}$$

Range constraints of the form $l \leq c(x) \leq u$ can be expressed in this formulation by introducing slack variables $x_s \geq 0$, $x_t \geq 0$ (increasing n by 2) and defining new equality constraints $g(x, x_s) \equiv c(x) - l - x_s = 0$ and $g(x, x_t) \equiv u - c(x) - x_t = 0$.

nag_opt_handle_solve_ipopt (e04stc), like the methods discussed in Williams and Lang (2013), Byrd *et al.* (2000), Conn *et al.* (2000) and Fiacco and McCormick (1990), computes (approximate) solutions for a sequence of barrier problems

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \varphi_\mu(x) f(x) - \mu \sum_{i=1}^{n} \ln\left(x^{(i)}\right) \tag{5}$$

$$\text{subject to} \quad g(x) = 0 \tag{6}$$

for a decreasing sequence of barrier parameters μ converging to zero.

The algorithm may be interpreted as a homotopy method to the primal-dual equations,

$$\nabla f(x) + \nabla g(x)\lambda - z = 0 \tag{7}$$

$$g(x) = 0 \tag{8}$$

$$XZe - \mu e = 0 \tag{9}$$

with the homotopy parameter μ, which is driven to zero (see e.g., Byrd *et al.* (1997) and Gould *et al.* (2001)). Here, $X\text{diag}(x)$ for a vector x (similarly $z\text{diag}(z)$, etc.), and e stands for the vector of all ones for appropriate dimension, while $\lambda \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ correspond to the Lagrange multipliers for the equality constraints (3) and the bound constraints (4), respectively.

Note, that the equations (7), (8) and (9) for $\mu = 0$ together with 'x, $z \geq 0$' are the Karush–Kuhn–Tucker (KKT) conditions for the original problem (2), (3) and (4). Those are the first order optimality conditions for (2), (3) and (4) if constraint qualifications are satisfied (Conn *et al.* (2000)).

Starting from an initial point supplied in **x**, nag_opt_handle_solve_ipopt (e04stc) computes an approximate solution to the barrier problem (5) and (6) for a fixed value of μ (by default, 0.1), then decreases the barrier parameter, and continues the solution of the next barrier problem from the approximate solution of the previous one.

A sophisticated overall termination criterion for the algorithm is used to overcome potential difficulties when the Lagrange multipliers become large. This can happen, for example, when the gradients of the active constraints are nearly linear dependent. The termination criterion is described in detail by Wìchter and Biegler (2006) (also see below Section 11.1).

11.1 Stopping Criteria

Using the individual parts of the primal-dual equations (7), (8) and (9), we define the optimality error for the barrier problem as

$$E_\mu(x,\lambda,z)\max\left\{\frac{\|\nabla f(x)+\nabla g(x)\lambda-z\|_\infty}{s_d},\|g(x)\|_\infty,\frac{\|XZe-\mu e\|_\infty}{s_c}\right\} \tag{10}$$

with scaling parameters $s_d,\ s_c \geq 1$ defined below (not to be confused with NLP scaling factors described in Section 11.2). By $E_0(x,\lambda,z)$ we denote (10) with $\mu=0$; this measures the optimality error for the original problem (2), (3) and (4). The overall algorithm terminates if an approximate solution $\left(\tilde{x}_*,\tilde{\lambda}_*,\tilde{z}_*\right)$ (including multiplier estimates) satisfying

$$E_0\left(\tilde{x}_*,\tilde{\lambda}_*,\tilde{z}_*\right)\leq \epsilon_{tol} \tag{11}$$

is found, where $\epsilon_{tol}>0$ is the user provided error tolerance in optional parameter **Stop Tolerance 1**.

Even if the original problem is well scaled, the multipliers λ and z might become very large, for example, when the gradients of the active constraints are (nearly) linearly dependent at a solution of (2), (3) and (4). In this case, the algorithm might encounter numerical difficulties satisfying the unscaled primal-dual equations (7), (8) and (9) to a tight tolerance. In order to adapt the termination criteria to handle such circumstances, we choose the scaling factors

$$s_d\max\left\{s_{\max},\frac{\|\lambda\|_1+\|z\|_1}{(m+n)}\right\}/s_{\max} \qquad s_c\max\left\{s_{\max},\frac{\|z\|_1}{n}\right\}/s_{\max}$$

in (10). In this way, a component of the optimality error is scaled, whenever the average value of the multipliers becomes larger than a fixed number $s_{\max}\geq 1$ ($s_{\max}=100$ in our implementation). Also note, in the case that the multipliers diverge, $E_0(x,\lambda,z)$ can only become small, if a Fritz John point for (2), (3) and (4) is approached, or if the primal variables diverge as well.

11.2 Scaling the NLP

Ideally, the formulated problem should be scaled so that, near the solution, all function gradients (objective and constraints), when nonzero, are of a similar order of a magnitude. nag_opt_handle_solve_ipopt (e04stc) will compute automatic NLP scaling factors for the objective and constraint functions (but not the decision variables) and apply them if large imbalances of scale are detected. This rescaling is only computed at the starting point. References to scaled or unscaled objective or constraints in Section 9.1 and Section 11 should be understood in this context.

12 Optional Parameters

Several optional parameters in nag_opt_handle_solve_ipopt (e04stc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_opt_handle_solve_ipopt (e04stc) 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
Hessian Mode
Infinite Bound Size
Monitoring File
Monitoring Level
Outer Iteration Limit
Print File
Print Level
Stats Time
Stop Tolerance 1
Time Limit
Verify Derivatives

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.

Hessian Mode a Default $=$ AUTO

This argument specifies whether the Hessian will be supplied by the user (in **hx**) or approximated by nag_opt_handle_solve_ipopt (e04stc) using a limited-memory quasi-Newton L-BFGS method. In the AUTO setting, if no Hessian structure has been registered in the problem with a call to nag_opt_handle_set_nlnhess (e04rlc), and there are explicitly nonlinear user-supplied functions, then the Hessian will be approximated. Otherwise **hess** will be called if and only if any of nag_opt_handle_set_nlnobj (e04rgc) or nag_opt_handle_set_nlnconstr (e04rkc) have been used to define the problem. Approximating the Hessian is likely to require more iterations to achieve convergence but will reduce the time spent in user-supplied functions.

Constraint: **Hessian Mode** $=$ AUTO, EXACT or APPROXIMATE.

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.

It also serves as a limit for the objective function to be considered unbounded (**fail.code** = NE_MAYBE_UNBOUNDED).

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

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 information output to this unit is controlled 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 iterations to be performed by nag_opt_handle_solve_ipopt (e04stc). Setting the option too low might lead to **fail.code** = NE_TOO_MANY_ITER.

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

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 information output to this unit is controlled 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 (except a one-time banner)
1	Additionally, derivative check information, the Header and Summary.
2	Additionally, the Iteration log.
3, 4	Additionally, details of each iteration with scalar quantities printed.
5	Additionally, individual components of arrays are printed resulting in large output.

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

Stats Time a Default $=$ NO

This argument allows you to turn 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.

Constraint: **Stats Time** $=$ YES or NO.

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

This option sets the value ϵ_{tol} which is used for optimality and complementarity tests from KKT conditions See Section 11.1.

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

Time Limit r Default $= 10^6$

A limit on seconds that the solver can use to solve one problem. If during the convergence check this limit is exceeded, the solver will terminate with a corresponding error message.

Constraint: **Time Limit** > 0.

Verify Derivatives a Default $=$ AUTO

This argument specifies whether the function should perform numerical checks on the consistency of the user-supplied functions. It is recommended that such checks are enabled when first developing the formulation of the problem. Option AUTO will perform the checks unless it is determined that there are no explicitly nonlinear user-supplied functions.

Constraint: **Verify Derivatives** $=$ AUTO, YES or NO.