NAG Library Function Document

nag_opt_nlp_revcomm (e04ufc)

Note: *this function uses* **optional arguments** *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 arguments, 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, to Section 12 for a detailed description of the specification of the optional arguments and to Section 13 for a detailed description of the monitoring information produced by the function.*

1 Purpose

nag_opt_nlp_revcomm (e04ufc) is designed to minimize an arbitrary smooth function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using a sequential quadratic programming (SQP) method. You should supply as many first derivatives as possible; any unspecified derivatives are approximated by finite differences. It is not intended for large sparse problems.

nag_opt_nlp_revcomm (e04ufc) may also be used for unconstrained, bound-constrained and linearly constrained optimization.

nag_opt_nlp_revcomm (e04ufc) uses **reverse communication** for evaluating the objective function, the nonlinear constraint functions and any of their derivatives.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_nlp_revcomm (Integer *irevcm, Integer n, Integer nclin,
     Integer ncnln, Integer tda, Integer tdcj, Integer tdr, const double a[],
     const double bl[], const double bu[], Integer *iter, Integer istate[],
     double c[], double cjac[], double clamda[], double *objf,
     double objgrd[], double r[], double x[], Integer needc[],
     Integer iwork[], Integer liwork, double work[], Integer lwork,
     char cwsav[], Nag_Boolean lwsav[], Integer iwsav[], double rwsav[],
     NagError *fail)
```

Before calling nag_opt_nlp_revcomm (e04ufc) or either of the option setting functions nag_opt_nlp_revcomm_option_set_file (e04udc) or nag_opt_nlp_revcomm_option_set_string (e04uec), nag_opt_nlp_revcomm_init (e04wbc) **must** be called. The specification for nag_opt_nlp_revcomm_init (e04wbc) is:

```
#include <nag.h>
#include <nage04.h>
void nag_opt_nlp_revcomm_init_b (const char *rname, Nag_E04StateB *state,
     NagError *fail)
```

nag_opt_nlp_revcomm_init (e04wbc) should be called with **rname** = 'e04ufc'. **cwsav** must have a length of **lcwsav** $\times$ 80, while the lengths of **lwsav**, **iwsav** and **rwsav** should be **llwsav**, **liwsav** and **lrwsav**, respectively. These arguments must satisfy the following constraints:

lcwsav $\geq$ 5

llwsav $\geq$ 120

liwsav $\geq$ 610

lrwsav $\geq$ 475

The contents of the arrays **cwsav**, **lwsav**, **iwsav** and **rwsav** MUST NOT be altered between calling functions nag_opt_nlp_revcomm_option_set_file (e04udc), nag_opt_nlp_revcomm_option_set_string (e04uec), nag_opt_nlp_revcomm (e04ufc) or nag_opt_nlp_revcomm_init (e04wbc).

3 Description

nag_opt_nlp_revcomm (e04ufc) is designed to solve the nonlinear programming problem – the minimization of a smooth nonlinear function subject to a set of constraints on the variables. The problem is assumed to be stated in the following form:

$$\underset{x \in R^n}{\text{minimize}}\ F(x) \quad \text{subject to} \quad l \leq \begin{pmatrix} x \\ A_L x \\ c(x) \end{pmatrix} \leq u, \tag{1}$$

where $F(x)$ (the *objective function*) is a nonlinear function, A_L is an n_L by n constant matrix, and $c(x)$ is an n_N element vector of nonlinear constraint functions. (The matrix A_L and the vector $c(x)$ may be empty.) The objective function and the constraint functions are assumed to be smooth, i.e., at least twice-continuously differentiable. (The method of nag_opt_nlp_revcomm (e04ufc) will usually solve (1) if there are only isolated discontinuities away from the solution.)

Note that although the bounds on the variables could be included in the definition of the linear constraints, we prefer to distinguish between them for reasons of computational efficiency. For the same reason, the linear constraints should **not** be included in the definition of the nonlinear constraints. Upper and lower bounds are specified for all the variables and for all the constraints. An *equality* constraint can be specified by setting $l_i = u_i$. If certain bounds are not present, the associated elements of l or u can be set to special values that will be treated as $-\infty$ or $+\infty$. (See the description of the optional argument **Infinite Bound Size**.)

If there are no nonlinear constraints in (1) and F is linear or quadratic then it will generally be more efficient to use one of nag_opt_lp (e04mfc), nag_opt_lin_lsq (e04ncc) or nag_opt_qp (e04nfc), or nag_opt_sparse_convex_qp_solve (e04nqc) if the problem is large and sparse. If the problem is large and sparse and does have nonlinear constraints, nag_opt_nlp_sparse (e04ugc) should be used, since nag_opt_nlp_revcomm (e04ufc) treats all matrices as dense.

nag_opt_nlp_revcomm (e04ufc) uses reverse communication for evaluating $F(x)$, $c(x)$ and as many of their first partial derivatives as possible; any remaining derivatives are approximated by finite differences. See the description of the optional argument **Derivative Level**.

On initial entry, you must supply an initial estimate of the solution to (1).

On intermediate exits, the calling program must compute appropriate values for the objective function, the nonlinear constraints or their derivatives, as specified by the argument **irevcm**, and then re-enter the function.

For maximum reliability, it is preferable to provide all partial derivatives (see Chapter 8 of Gill *et al.* (1981), for a detailed discussion). If they cannot all be provided, it is advisable to provide as many as possible. While developing code to evaluate the objective function and the constraints, the optional argument **Verify** should be used to check the calculation of any known derivatives.

The method used by nag_opt_nlp_revcomm (e04ufc) is described in detail in Section 11.

nag_opt_nlp (e04ucc) and nag_opt_nlp_solve (e04wdc) are alternative functions which use a similar method, but with **direct communicaiton**: that is, the objective and constraint functions are evaluated by functions, supplied as arguments to the function.

4 References

Dennis J E Jr and Moré J J (1977) Quasi-Newton methods, motivation and theory *SIAM Rev.* **19** 46–89

Dennis J E Jr and Schnabel R B (1981) A new derivation of symmetric positive-definite secant updates *nonlinear programming* (eds O L Mangasarian, R R Meyer and S M Robinson) **4** 167–199 Academic Press

Dennis J E Jr and Schnabel R B (1983) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* Prentice–Hall

Fletcher R (1987) *Practical Methods of Optimization* (2nd Edition) Wiley

Gill P E, Hammarling S, Murray W, Saunders M A and Wright M H (1986) Users' guide for LSSOL (Version 1.0) *Report SOL 86-1* Department of Operations Research, Stanford University

Gill P E, Murray W, Saunders M A and Wright M H (1984a) Procedures for optimization problems with a mixture of bounds and general linear constraints *ACM Trans. Math. Software* **10** 282–298

Gill P E, Murray W, Saunders M A and Wright M H (1984b) Users' guide for SOL/QPSOL version 3.2 *Report SOL* 84–5 Department of Operations Research, Stanford University

Gill P E, Murray W, Saunders M A and Wright M H (1986a) Some theoretical properties of an augmented Lagrangian merit function *Report SOL* 86–6R Department of Operations Research, Stanford University

Gill P E, Murray W, Saunders M A and Wright M H (1986b) Users' guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming *Report SOL 86-2* Department of Operations Research, Stanford University

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

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

Murtagh B A and Saunders M A (1983) MINOS 5.0 user's guide *Report SOL 83-20* Department of Operations Research, Stanford University

Powell M J D (1974) Introduction to constrained optimization *Numerical Methods for Constrained Optimization* (eds P E Gill and W Murray) 1–28 Academic Press

Powell M J D (1983) Variable metric methods in constrained optimization *Mathematical Programming: the State of the Art* (eds A Bachem, M Grötschel and B Korte) 288–311 Springer–Verlag

5 Arguments

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than those specified by the value of irevcm must remain unchanged**.

1: **irevcm** – Integer * *Input/Output*

On initial entry: must be set to 0.

On intermediate exit: specifies what values the calling program must assign to arguments of nag_opt_nlp_revcomm (e04ufc) before re-entering the function.

irevcm $= 1$
Set **objf** to the value of the objective function $F(x)$.

irevcm $= 2$
Set **objgrd**$[j-1]$ to the value $\dfrac{\partial F}{\partial x_j}$ if available, for $j = 1, 2, \ldots, n$.

irevcm $= 3$
Set **objf** and **objgrd**$[j-1]$ as for **irevcm** $= 1$ and **irevcm** $= 2$.

irevcm $= 4$
Set **c**$[i-1]$ to the value of the constraint function $c_i(x)$, for each i such that **needc**$[i-1] > 0$.

irevcm $= 5$
Set **CJAC**(i, j) to the value $\dfrac{\partial c_i}{\partial x_j}$ if available, for each i such that **needc**$[i-1] > 0$ and $j = 1, 2, \ldots, n$, where **CJAC**(i, j) is as defined in **cjac**.

irevcm $= 6$
Set $\mathbf{CJAC}(i,j)$ as for **irevcm** $= 4$ and **irevcm** $= 5$.

On intermediate re-entry: **must remain unchanged**, unless you wish to terminate the solution to the current problem. In this case **irevcm** may be set to a negative value and then nag_opt_nlp_revcomm (e04ufc) will take a final exit with **fail.code** = NE_USER_STOP.

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

Constraint: **irevcm** ≤ 6.

2: **n** – Integer *Input*

On initial entry: n, the number of variables.

Constraint: **n** > 0.

3: **nclin** – Integer *Input*

On initial entry: n_L, the number of general linear constraints.

Constraint: **nclin** ≥ 0.

4: **ncnln** – Integer *Input*

On initial entry: n_N, the number of nonlinear constraints.

Constraint: **ncnln** ≥ 0.

5: **tda** – Integer *Input*

On entry: the stride separating matrix column elements in the array **a**.

Constraints:

if **nclin** > 0, **tda** $\geq$ **n**;
otherwise **tda** ≥ 1.

6: **tdcj** – Integer *Input*

On entry: the stride separating matrix column elements in the array **cjac**.

Constraints:

if **ncnln** > 0, **tdcj** $\geq$ **n**;
otherwise **tdcj** ≥ 1.

7: **tdr** – Integer *Input*

On entry: the stride separating matrix column elements in the array **r**.

Constraint: **tdr** $\geq$ **n**.

8: **a**$[dim]$ – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{nclin} \times \mathbf{tda})$.

Where $\mathbf{A}(i,j)$ appears in this document, it refers to the array element $\mathbf{a}[(i-1) \times \mathbf{tda} + j - 1]$.

On initial entry: the ith row of the matrix A_L of general linear constraints in (1) must be stored in $\mathbf{A}(i,j)$, for $i = 1, 2, \ldots,$ **nclin** and $j = 1, 2, \ldots,$ **n**. That is, the ith row contains the coefficients of the ith general linear constraint, for $i = 1, 2, \ldots,$ **nclin**.

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

9: **bl**[**n** + **nclin** + **ncnln**] – const double *Input*
10: **bu**[**n** + **nclin** + **ncnln**] – const double *Input*

On initial entry: **bl** must contain the lower bounds and **bu** the upper bounds, for all the constraints in the following order. The first n elements of each array must contain the bounds on the variables, the next n_L elements the bounds for the general linear constraints (if any) and the next n_N elements the bounds for the general nonlinear constraints (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set **bl**$[j-1] \leq -bigbnd$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set **bu**$[j-1] \geq bigbnd$; the default value of $bigbnd$ is 10^{20}, but this may be changed by the optional argument **Infinite Bound Size**. To specify the jth constraint as an *equality*, set **bl**$[j-1] =$ **bu**$[j-1] = \beta$, say, where $|\beta| < bigbnd$.

Constraints:

bl$[j-1] \leq$ **bu**$[j-1]$, for $j = 1, 2, \ldots,$ **n** + **nclin** + **ncnln**;
if **bl**$[j] =$ **bu**$[j] = \beta$, $|\beta| < bigbnd$.

11: **iter** – Integer * *Input/Output*

On intermediate re-entry: must remain unchanged from a previous call to nag_opt_nlp_revcomm (e04ufc).

On final exit: the number of major iterations performed.

12: **istate**[**n** + **nclin** + **ncnln**] – Integer *Input/Output*

On initial entry: need not be set if the (default) optional argument **Cold Start** is used.

If the optional argument **Warm Start** has been chosen, the elements of **istate** corresponding to the bounds and linear constraints define the initial working set for the procedure that finds a feasible point for the linear constraints and bounds. The active set at the conclusion of this procedure and the elements of **istate** corresponding to nonlinear constraints then define the initial working set for the first QP subproblem. More precisely, the first n elements of **istate** refer to the upper and lower bounds on the variables, the next n_L elements refer to the upper and lower bounds on $A_L x$, and the next n_N elements refer to the upper and lower bounds on $c(x)$. Possible values for **istate**$[j-1]$ are as follows:

istate$[j-1]$	**Meaning**
0	The corresponding constraint is *not* in the initial QP working set.
1	This inequality constraint should be in the working set at its lower bound.
2	This inequality constraint should be in the working set at its upper bound.
3	This equality constraint should be in the initial working set. This value must not be specified unless **bl**$[j-1] =$ **bu**$[j-1]$.

The values -2, -1 and 4 are also acceptable but will be modified by the function. If nag_opt_nlp_revcomm (e04ufc) has been called previously with the same values of **n**, **nclin** and **ncnln**, **istate** already contains satisfactory information. (See also the description of the optional argument **Warm Start**.) The function also adjusts (if necessary) the values supplied in **x** to be consistent with **istate**.

Constraint: $-2 \leq$ **istate**$[j-1] \leq 4$, for $j = 1, 2, \ldots,$ **n** + **nclin** + **ncnln**.

On final exit: the status of the constraints in the QP working set at the point returned in **x**. The significance of each possible value of **istate**$[j-1]$ is as follows:

istate$[j-1]$	**Meaning**
-2	This constraint violates its lower bound by more than the appropriate feasibility tolerance (see the optional arguments **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance**). This value can occur only when no feasible point can be found for a QP subproblem.

-1	This constraint violates its upper bound by more than the appropriate feasibility tolerance (see the optional arguments **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance**). This value can occur only when no feasible point can be found for a QP subproblem.
0	The constraint is satisfied to within the feasibility tolerance, but is not in the QP working set.
1	This inequality constraint is included in the QP working set at its lower bound.
2	This inequality constraint is included in the QP working set at its upper bound.
3	This constraint is included in the QP working set as an equality. This value of **istate** can occur only when $\mathbf{bl}[j-1] = \mathbf{bu}[j-1]$.

13: **c**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **c** must be at least $\max(1, \mathbf{ncnln})$.

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 4$ or 6 and **needc**$[i-1] > 0$, **c**$[i-1]$ must contain the value of the ith constraint at x. The remaining elements of **c**, corresponding to the non-positive elements of **needc**, are ignored.

On final exit: if **ncnln** > 0, **c**$[i-1]$ contains the value of the ith nonlinear constraint function c_i at the final iterate, for $i = 1, 2, \ldots,$ **ncnln**.

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

14: **cjac**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **cjac** must be at least $\max(1, \mathbf{ncnln} \times \mathbf{tdcj})$.

Where **CJAC**(i, j) appears in this document, it refers to the array element **cjac**$[(i-1) \times \mathbf{tdcj} + j - 1]$.

On initial entry: in general, **cjac** need not be initialized before the call to nag_opt_nlp_revcomm (e04ufc). However, if the optional argument **Derivative Level** $= 2$ or 3, you may optionally set the constant elements of **cjac**. Such constant elements need not be re-assigned on subsequent intermediate exits.

If all elements of the constraint Jacobian are known (i.e., **Derivative Level** $= 2$ or 3), any constant elements may be assigned to **cjac** one time only at the start of the optimization. An element of **cjac** that is not subsequently assigned during an intermediate exit will retain its initial value throughout. Constant elements may be loaded into **cjac** either before the call to nag_opt_nlp_revcomm (e04ufc) or during the first intermediate exit. The ability to preload constants is useful when many Jacobian elements are identically zero, in which case **cjac** may be initialized to zero and nonzero elements may be reset during intermediate exits.

On intermediate re-entry: if **irevcm** $= 5$ or 6 and **needc**$[i-1] > 0$, the ith row of **CJAC** must contain the available elements of the vector ∇c_i given by

$$\nabla c_i = \left(\frac{\partial c_i}{\partial x_1}, \frac{\partial c_i}{\partial x_2}, \ldots, \frac{\partial c_i}{\partial x_n}\right)^{\mathrm{T}},$$

where $\dfrac{\partial c_i}{\partial x_j}$ is the partial derivative of the ith constraint with respect to the jth variable, evaluated at the point x. The remaining rows of **CJAC**, corresponding to non-positive elements of **needc**, are ignored. The ith row of the Jacobian should be stored in elements **CJAC**(i, j), for $i = 1, 2, \ldots,$ **ncnln** and $j = 1, 2, \ldots,$ **n**.

Note that constant nonzero elements do affect the values of the constraints. Thus, if **CJAC**(i, j) is set to a constant value, it need not be reset during subsequent intermediate exits, but the value **CJAC**$(i, j) \times$ **x**$[j-1]$ must nonetheless be added to **c**$[i-1]$. For example, if **CJAC**$(1, 1) = 2$ and **CJAC**$(1, 2) = -5$, then the term $2 \times \mathbf{x}[0] - 5 \times \mathbf{x}[1]$ must be included in the definition of **c**$[0]$.

It must be emphasized that, if **Derivative Level** $= 0$ or 1, unassigned elements of **cjac** are not treated as constant; they are estimated by finite differences, at nontrivial expense. If you do not supply a value for the optional argument **Difference Interval**, an interval for each element of x is computed automatically at the start of the optimization. The automatic procedure can usually identify constant elements of **cjac**, which are then computed once only by finite differences.

See also the description of the optional argument **Verify**.

On final exit: if **ncnln** > 0, **cjac** contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., **CJAC**(i,j) contains the partial derivative of the ith constraint function with respect to the jth variable, for $i = 1, 2, \ldots,$ **ncnln** and $j = 1, 2, \ldots,$ **n**.

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

15: **clamda**[**n** + **nclin** + **ncnln**] – double *Input/Output*

On initial entry: need not be set if the (default) optional argument **Cold Start** is used.

If the optional argument **Warm Start** has been chosen, **clamda**$[j-1]$ must contain a multiplier estimate for each nonlinear constraint with a sign that matches the status of the constraint specified by the **istate** array, for $j =$ **n** + **nclin** $+ 1, \ldots,$ **n** + **nclin** + **ncnln**. The remaining elements need not be set. Note that if the jth constraint is defined as 'inactive' by the initial value of the **istate** array (i.e. **istate**$[j-1] = 0$), **clamda**$[j-1]$ should be zero; if the jth constraint is an inequality active at its lower bound (i.e. **istate**$[j-1] = 1$), **clamda**$[j-1]$ should be non-negative; if the jth constraint is an inequality active at its upper bound (i.e. **istate**$[j-1] = 2$), **clamda**$[j-1]$ should be non-positive. If necessary, the function will modify **clamda** to match these rules.

On final exit: the values of the QP multipliers from the last QP subproblem. **clamda**$[j-1]$ should be non-negative if **istate**$[j-1] = 1$ and non-positive if **istate**$[j-1] = 2$.

16: **objf** – double * *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 1$ or 3, **objf** must be set to the value of the objective function at x.

On final exit: the value of the objective function at the final iterate.

17: **objgrd**[**n**] – double *Input/Output*

On initial entry: need not be set.

On intermediate re-entry: if **irevcm** $= 2$ or 3, **objgrd** must contain the available elements of the gradient evaluated at x.

See also the description of the optional argument **Verify**.

On final exit: the gradient of the objective function at the final iterate (or its finite difference approximation).

18: **r**[**n** × **tdr**] – double *Input/Output*

Note: the (i,j)th element of the matrix R is stored in **r**$[(i-1) \times$ **tdr** $+ j - 1]$.

On initial entry: need not be initialized if the (default) optional argument **Cold Start** is used.

If the optional argument **Warm Start** has been chosen, **r** must contain the upper triangular Cholesky factor R of the initial approximation of the Hessian of the Lagrangian function, with the variables in the natural order. Elements not in the upper triangular part of **r** are assumed to be zero and need not be assigned.

On final exit: if **Hessian** = NO, **r** contains the upper triangular Cholesky factor R of $Q^{\mathrm{T}}\tilde{H}Q$, an estimate of the transformed and reordered Hessian of the Lagrangian at x (see (6) in Section 11.1).

If **Hessian** = YES, **r** contains the upper triangular Cholesky factor R of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

19: **x**[**n**] – double *Input/Output*

On initial entry: an initial estimate of the solution.

On intermediate exit: the point x at which the objective function, constraint functions or their derivatives are to be evaluated.

On final exit: the final estimate of the solution.

20: **needc**[**max**(**1**, **ncnln**)] – Integer *Output*

On intermediate exit: if **irevcm** ≥ 4, **needc** specifies the indices of the elements of **c** and/or **cjac** that must be assigned. If **needc**$[i-1] > 0$, then the ith element of **c** and/or the available elements of the ith row of **cjac** must be evaluated at x.

21: **iwork**[**liwork**] – Integer *Communication Array*

22: **liwork** – Integer *Input*

On initial entry: the dimension of the array **iwork**.

Constraint: **liwork** $\geq 3 \times$ **n** + **nclin** $+ 2 \times$ **ncnln**.

23: **work**[**lwork**] – double *Communication Array*

24: **lwork** – Integer *Input*

On initial entry: the dimension of the array **work**.

Constraints:

if **ncnln** $= 0$ and **nclin** $= 0$, **lwork** $\geq 21 \times$ **n** $+ 2$;
if **ncnln** $= 0$ and **nclin** > 0, **lwork** $\geq 2 \times$ **n**$^2 + 21 \times$ **n** $+ 11 \times$ **nclin** $+ 2$;
if **ncnln** > 0 and **nclin** ≥ 0,
lwork $\geq 2 \times$ **n**$^2 +$ **n** $\times$ **nclin** $+ 2 \times$ **n** $\times$ **ncnln** $+ 21 \times$ **n** $+ 11 \times$ **nclin** $+$
$22 \times$ **ncnln** $+ 1$.

The contents of the communication arrays **iwork** and **work** must not be altered between calls to nag_opt_nlp_revcomm (e04ufc).

25: **cwsav**[400] – char *Communication Array*

26: **lwsav**[**120**] – Nag_Boolean *Communication Array*

27: **iwsav**[**610**] – Integer *Communication Array*

28: **rwsav**[**475**] – double *Communication Array*

The arrays **lwsav**, **iwsav**, **rwsav** and **cwsav** MUST NOT be altered between calls to any of the functions nag_opt_nlp_revcomm_init (e04wbc), nag_opt_nlp_revcomm_option_set_file (e04udc), nag_opt_nlp_revcomm_option_set_string (e04uec) or nag_opt_nlp_revcomm (e04ufc).

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_BOUND

On entry, the bounds on linear constraint $\langle value \rangle$ are inconsistent: $\mathbf{bl}[\langle value \rangle] = \langle value \rangle$ and $\mathbf{bu}[\langle value \rangle] = \langle value \rangle$.

On entry, the bounds on nonlinear constraint $\langle value \rangle$ are inconsistent: $\mathbf{bl}[\langle value \rangle] = \langle value \rangle$ and $\mathbf{bu}[\langle value \rangle] = \langle value \rangle$.

On entry, the bounds on variable $\langle value \rangle$ are inconsistent: $\mathbf{bl}[\langle value \rangle] = \langle value \rangle$ and $\mathbf{bu}[\langle value \rangle] = \langle value \rangle$.

NE_BOUND_EQ

On entry, the equal bounds on linear constraint $\langle value \rangle$ are infinite, because $\mathbf{bl}[\langle value \rangle] =$ beta and $\mathbf{bu}[\langle value \rangle] =$ beta, but $|\text{beta}| \geq$ bigbnd: beta $= \langle value \rangle$ and bigbnd $= \langle value \rangle$.

On entry, the equal bounds on nonlinear constraint $\langle value \rangle$ are infinite, because $\mathbf{bl}[\langle value \rangle] =$ beta and $\mathbf{bu}[\langle value \rangle] =$ beta, but $|\text{beta}| \geq$ bigbnd: beta $= \langle value \rangle$ and bigbnd $= \langle value \rangle$.

On entry, the equal bounds on variable $\langle value \rangle$ are infinite, because $\mathbf{bl}[\langle value \rangle] =$ beta and $\mathbf{bu}[\langle value \rangle] =$ beta, but $|\text{beta}| \geq$ bigbnd: beta $= \langle value \rangle$ and bigbnd $= \langle value \rangle$.

NE_DERIV_ERRORS

Large errors found in the derivatives.

The user-supplied derivatives of the objective function and/or nonlinear constraints appear to be incorrect.

*Large errors were found in the derivatives of the objective function and/or nonlinear constraints. This value of **fail** will occur if the verification process indicated that at least one gradient or Jacobian element had no correct figures. You should refer to the printed output to determine which elements are suspected to be in error.*

As a first-step, you should check that the code for the objective and constraint values is correct – for example, by computing the function at a point where the correct value is known. However, care should be taken that the chosen point fully tests the evaluation of the function. It is remarkable how often the values $x = 0$ or $x = 1$ are used in such a test, and how often the special properties of these numbers make the test meaningless.

Gradient checking will be ineffective if the objective function uses information computed by the constraints, since they are not necessarily computed before each function evaluation.

Errors in programming the function may be quite subtle in that the function value is 'almost' correct. For example, the function may not be accurate to full precision because of the inaccurate calculation of a subsidiary quantity, or the limited accuracy of data upon which the function depends. A common error on machines where numerical calculations are usually performed in double precision is to include even one single precision constant in the calculation of the function; since some compilers do not convert such constants to double precision, half the correct figures may be lost by such a seemingly trivial error.

NE_INT

On entry, **irevcm** $= \langle value \rangle$.
Constraint: $6 \geq$ **irevcm** ≥ 0.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** > 0.

On entry, **nclin** $= \langle value \rangle$.
Constraint: **nclin** ≥ 0.

On entry, **ncnln** $= \langle value \rangle$.
Constraint: **ncnln** ≥ 0.

NE_INT_2

On entry, **tdr** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **tdr** $\geq$ **n**.

NE_INT_3

On entry, **tda** $= \langle value \rangle$, **nclin** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **nclin** > 0, **tda** $\geq$ **n**;
otherwise **tda** ≥ 1.

On entry, **tdcj** $= \langle value \rangle$, **ncnln** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **ncnln** > 0, **tdcj** $\geq$ **n**;
otherwise **tdcj** ≥ 1.

NE_INT_4

On entry, **n** $= \langle value \rangle$, **nclin** $= \langle value \rangle$, **ncnln** $= \langle value \rangle$ and **liwork** $= \langle value \rangle$.
Constraint: **liwork** $\geq 3 \times$ **n** $+$ **nclin** $+ 2 \times$ **ncnln**.

On entry, **n** $= \langle value \rangle$, **nclin** $= \langle value \rangle$, **ncnln** $= \langle value \rangle$ and **lwork** $= \langle value \rangle$.
Constraint: if **ncnln** $= 0$ and **nclin** $= 0$, **lwork** $\geq 21 \times$ **n** $+ 2$.

On entry, **n** $= \langle value \rangle$, **nclin** $= \langle value \rangle$, **ncnln** $= \langle value \rangle$ and **lwork** $= \langle value \rangle$.
Constraint: if **ncnln** $= 0$ and **nclin** > 0, **lwork** $\geq 2 \times$ **n**$^2 + 21 \times$ **n** $+ 11 \times$ **nclin** $+ 2$.

On entry, **n** $= \langle value \rangle$, **nclin** $= \langle value \rangle$, **ncnln** $= \langle value \rangle$ and **lwork** $= \langle value \rangle$.
Constraint: if **ncnln** > 0 and **nclin** ≥ 0,
lwork $\geq 2 \times$ **n**$^2 +$ **n** $\times$ **nclin** $+ 2 \times$ **n** $\times$ **ncnln** $+ 21 \times$ **n** $+ 11 \times$ **nclin** $+$
$22 \times$ **ncnln** $+ 1$.

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 the Essential Introduction for further information.

NE_LIN_NOT_FEASIBLE

No feasible point for the linear constraints.

nag_opt_nlp_revcomm (e04ufc) has terminated without finding a feasible point for the linear constraints and bounds, which means that either no feasible point exists for the given value of the optional argument **Linear Feasibility Tolerance**, *or no feasible point could be found in the number of iterations specified by the optional argument* **Minor Iteration Limit**. *You should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ, you should ensure that the value of the optional argument*
Linear Feasibility Tolerance *is greater than σ. For example, if all elements of A_L are of order unity and are accurate to only three decimal places,* **Linear Feasibility Tolerance** *should be at least 10^{-3}.*

NE_NO_IMPROVEMENT

Current point cannot be improved upon.

x does not satisfy the first-order Kuhn–Tucker conditions (see Section 11.1), and no improved point for the merit function (see Section 9.1) could be found during the final linesearch.

This sometimes occurs because an overly stringent accuracy has been requested, i.e., the value of the optional argument **Optimality Tolerance** *(default value $= \epsilon_r^{0.8}$, where ϵ_r is the value of the optional argument* **Function Precision***) is too small. In this case you should apply the four tests described under* **fail**.**code** $=$ *NE_NOERROR to determine whether or not the final solution is acceptable (see Gill et al. (1981), for a discussion of the attainable accuracy).*

If many iterations have occurred in which essentially no progress has been made and nag_opt_nlp_revcomm (e04ufc) has failed completely to move from the initial point, then values set by the calling program for the objective or constraint functions or their derivatives during intermediate exits may be incorrect. You should refer to comments under **fail**.**code** = *NE_DERIV_ERRORS and check the gradients using the optional argument* **Verify**. *Unfortunately, there may be small errors in the objective and constraint gradients that cannot be detected by the verification process. Finite difference approximations to first derivatives are catastrophically affected by even small inaccuracies. An indication of this situation is a dramatic alteration in the iterates if the finite difference interval is altered. One might also suspect this type of error if a switch is made to central differences even when* `Norm Gz` *and* `Violtn` *(see Section 9.1) are large.*

Another possibility is that the search direction has become inaccurate because of ill-conditioning in the Hessian approximation or the matrix of constraints in the working set; either form of ill-conditioning tends to be reflected in large values of `Mnr` *(the number of iterations required to solve each QP subproblem; see Section 9.1).*

If the condition estimate of the projected Hessian (`Cond Hz`*; see Section 9.1) is extremely large, it may be worthwhile rerunning nag_opt_nlp_revcomm (e04ufc) from the final point with the optional argument* **Warm Start**. *In this situation,* ***istate*** *and* ***clamda*** *should be left unaltered and* **r** *should be reset to the identity matrix.*

If the matrix of constraints in the working set is ill-conditioned (i.e., `Cond T` *is extremely large; see Section 13), it may be helpful to run nag_opt_nlp_revcomm (e04ufc) with a relaxed value of the optional argument* **Feasibility Tolerance**. *(Constraint dependencies are often indicated by wide variations in size in the diagonal elements of the matrix T, whose diagonals will be printed if* **Major Print Level** $\geq$ *30.)*

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_NONLIN_NOT_FEASIBLE

No feasible point for the nonlinear constraints.

No feasible point could be found for the nonlinear constraints. The problem may have no feasible solution. This means that there has been a sequence of QP subproblems for which no feasible point could be found (indicated by `I` *at the end of each line of intermediate printout produced by the major iterations; see Section 9.1). This behaviour will occur if there is no feasible point for the nonlinear constraints. (However, there is no general test that can determine whether a feasible point exists for a set of nonlinear constraints.) If the infeasible subproblems occur from the very first major iteration, it is highly likely that no feasible point exists. If infeasibilities occur when earlier subproblems have been feasible, small constraint inconsistencies may be present. You should check the validity of constraints with negative values of* ***istate***. *If you are convinced that a feasible point does exist, nag_opt_nlp_revcomm (e04ufc) should be restarted at a different starting point.*

NE_TOO_MANY

Too many major iterations.

The limiting number of iterations (as determined by the optional argument **Major Iteration Limit***) has been reached.*

If the algorithm appears to be making satisfactory progress, then optional argument **Major Iteration Limit** *may be too small. If so, either increase its value and rerun nag_opt_nlp_revcomm (e04ufc) or, alternatively, rerun nag_opt_nlp_revcomm (e04ufc) using the optional argument* **Warm Start**. *If the algorithm seems to be making little or no progress however, then you should check for incorrect gradients or ill-conditioning as described under* **fail**.**code** = *NE_NO_IMPROVEMENT.*

Note that ill-conditioning in the working set is sometimes resolved automatically by the algorithm, in which case performing additional iterations may be helpful. However, ill-conditioning in the Hessian approximation tends to persist once it has begun, so that allowing additional iterations

without altering **r** *is usually inadvisable. If the quasi-Newton update of the Hessian approximation was reset during the latter major iterations (i.e., an* `R` *occurs at the end of each line of intermediate printout; see Section 9.1), it may be worthwhile to try a* **Warm Start** *at the final point as suggested above.*

NE_USER_STOP

User requested termination.

NE_WARM_START

On entry with a Warm Start, **istate**$[\langle value \rangle] = \langle value \rangle$.

NW_NOT_CONVERGED

Optimal solution found, but requested accuracy not achieved.

The final iterate x satisfies the first-order Kuhn–Tucker conditions (see Section 11.1) to the accuracy requested, but the sequence of iterates has not yet converged. nag_opt_nlp_revcomm (e04ufc) was terminated because no further improvement could be made in the merit function (see Section 9.1).

The most common situation is that you ask for a solution with accuracy that is not attainable with the given precision of the problem (as specified by the optional argument **Function Precision***). This condition will also occur if, by chance, an iterate is an 'exact' Kuhn–Tucker point, but the change in the variables was significant at the previous iteration. (This situation often happens when minimizing very simple functions, such as quadratics.)*

7 Accuracy

If **fail.code** = NE_NOERROR on final exit then the vector returned in the array **x** is an estimate of the solution to an accuracy of approximately **Optimality Tolerance** (default value $= \epsilon^{0.8}$, where ϵ is the ***machine precision***).

8 Parallelism and Performance

nag_opt_nlp_revcomm (e04ufc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_opt_nlp_revcomm (e04ufc) 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' Note for your implementation for any additional implementation-specific information.

9 Further Comments

9.1 Description of the Printed Output

This section describes the intermediate printout and final printout produced by nag_opt_nlp_revcomm (e04ufc). The intermediate printout is a subset of the monitoring information produced by nag_opt_nlp_revcomm (e04ufc) at every iteration (see Section 13). You can control the level of printed output (see the description of the optional argument **Major Print Level**). Note that the intermediate printout and final printout are produced only if **Major Print Level** ≥ 10 (by default no output is produced by nag_opt_nlp_revcomm (e04ufc).)

The following line of summary output (< 80 characters) is produced at every major iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Maj` is the major iteration count.

`Mnr` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 11).

Note that `Mnr` may be greater than the optional argument **Minor Iteration Limit** if some iterations are required for the feasibility phase.

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Merit Function` is the value of the augmented Lagrangian merit function (12) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty arguments (see Section 11.3). As the solution is approached, `Merit Function` will converge to the value of the objective function at the solution.

If the QP subproblem does not have a feasible point (signified by `I` at the end of the current output line) then the merit function is a large multiple of the constraint violations, weighted by the penalty arguments. During a sequence of major iterations with infeasible subproblems, the sequence of `Merit Function` values will decrease monotonically until either a feasible subproblem is obtained or nag_opt_nlp_revcomm (e04ufc) terminates with **fail.code** = NE_NONLIN_NOT_FEASIBLE (no feasible point could be found for the nonlinear constraints).

If there are no nonlinear constraints present (i.e., **ncnln** $= 0$) then this entry contains `Objective`, the value of the objective function $F(x)$. The objective function will decrease monotonically to its optimal value when there are no nonlinear constraints.

`Norm Gz` is $\|Z^{\mathrm{T}} g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 11.2). `Norm Gz` will be approximately zero in the neighbourhood of a solution.

`Violtn` is the Euclidean norm of the residuals of constraints that are violated or in the predicted active set (not printed if **ncnln** is zero). `Violtn` will be approximately zero in the neighbourhood of a solution.

`Cond Hz` is a lower bound on the condition number of the projected Hessian approximation H_Z ($H_Z = Z^{\mathrm{T}} H_{\mathrm{FR}} Z = R_Z^{\mathrm{T}} R_Z$; see (6)). The larger this number, the more difficult the problem.

`M` is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive definite (see Section 11.4).

`I` is printed if the QP subproblem has no feasible point.

`C` is printed if central differences have been used to compute the unspecified objective and constraint gradients. If the value of `Step` is zero then the switch to central differences was made because no lower point could be found in the linesearch. (In this case, the QP subproblem is resolved with the central difference gradient and Jacobian.) If the value of `Step` is nonzero then central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 11.1).

`L` is printed if the linesearch has produced a relative change in x greater than the value defined by the optional argument **Step Limit**. If this output occurs frequently during later iterations of the run, optional argument **Step Limit** should be set to a larger value.

`R` is printed if the approximate Hessian has been refactorized. If the diagonal condition estimator of R indicates that the approximate Hessian is badly conditioned then the approximate Hessian is refactorized using column

interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.

The final printout includes a listing of the status of every variable and constraint.

The following describes the printout for each variable. A full stop (.) is printed for any numerical value that is zero.

`Varbl` gives the name (`V`) and index j, for $j = 1, 2, \ldots, n$, of the variable.

`State` gives the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the **Feasibility Tolerance**, `State` will be `++` or `--` respectively.

A key is sometimes printed before `State`.

- `A` *Alternative optimum possible*. The variable is active at one of its bounds, but its Lagrange multiplier is essentially zero. This means that if the variable were allowed to start moving away from its bound then there would be no change to the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case the values of the Lagrange multipliers might also change.
- `D` *Degenerate*. The variable is free, but it is equal to (or very close to) one of its bounds.
- `I` *Infeasible*. The variable is currently violating one of its bounds by more than the **Feasibility Tolerance**.

`Value` is the value of the variable at the final iteration.

`Lower Bound` is the lower bound specified for the variable. `None` indicates that $\mathbf{bl}[j-1] \leq -bigbnd$.

`Upper Bound` is the upper bound specified for the variable. `None` indicates that $\mathbf{bu}[j-1] \geq bigbnd$.

`Lagr Mult` is the Lagrange multiplier for the associated bound. This will be zero if `State` is `FR` unless $\mathbf{bl}[j-1] \leq -bigbnd$ and $\mathbf{bu}[j-1] \geq bigbnd$, in which case the entry will be blank. If x is optimal, the multiplier should be non-negative if `State` is `LL` and non-positive if `State` is `UL`.

`Slack` is the difference between the variable `Value` and the nearer of its (finite) bounds $\mathbf{bl}[j-1]$ and $\mathbf{bu}[j-1]$. A blank entry indicates that the associated variable is not bounded (i.e., $\mathbf{bl}[j-1] \leq -bigbnd$ and $\mathbf{bu}[j-1] \geq bigbnd$).

The meaning of the printout for linear and nonlinear constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', $\mathbf{bl}[j-1]$ and $\mathbf{bu}[j-1]$ replaced by $\mathbf{bl}[n+j-1]$ and $\mathbf{bu}[n+j-1]$ respectively and with the following changes in the heading:

`L Con` gives the name (`L`) and index j, for $j = 1, 2, \ldots, n_L$, of the linear constraint.

`N Con` gives the name (`N`) and index $(j - n_L)$, for $j = n_L + 1, \ldots, n_L + n_N$, of the nonlinear constraint.

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

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

A full description of the printed output for minor iterations is given in Section 13.2.

10 Example

This is based on Problem 71 in Murtagh and Saunders (1983) and involves the minimization of the nonlinear function

$$F(x) = x_1x_4(x_1 + x_2 + x_3) + x_3$$

subject to the bounds

$$\begin{gathered} 1 \leq x_1 \leq 5 \\ 1 \leq x_2 \leq 5 \\ 1 \leq x_3 \leq 5 \\ 1 \leq x_4 \leq 5 \end{gathered}$$

to the general linear constraint

$$x_1 + x_2 + x_3 + x_4 \leq 20,$$

and to the nonlinear constraints

$$\begin{gathered} x_1^2 + x_2^2 + x_3^2 + x_4^2 \leq 40, \\ x_1x_2x_3x_4 \geq 25. \end{gathered}$$

The initial point, which is infeasible, is

$$x_0 = (1, 5, 5, 1)^{\mathrm{T}},$$

and $F(x_0) = 16$.

The optimal solution (to five figures) is

$$x^* = (1.0, 4.7430, 3.8211, 1.3794)^{\mathrm{T}},$$

and $F(x^*) = 17.014$. One bound constraint and both nonlinear constraints are active at the solution.

10.1 Program Text

```
/* nag_opt_nlp_revcomm (e04ufc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 *
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>

int main(void)
{
  /* Scalars */
  double        objf, nctotal;
  Integer       exit_status=0, i, irevcm, iter, j, n, nclin, ncnln;
  Integer       tda, tdcj, tdr, liwork, lwork;

  /* Arrays */
  double        *a=0, *bl=0, *bu=0, *c=0, *cjac=0, *clamda=0, *objgrd=0;
  double        *r=0, *work=0, rwsav[475], *x=0;
  Nag_Boolean   lwsav[120];
  Integer       *iwork=0, *istate=0, iwsav[610], *needc = 0;
  char          cwsav[5*80];

  /* Nag Types */
  NagError     fail;

#define A(I,J) a[(I-1)*tda + J - 1]
#define CJAC(I,J) cjac[(I-1)*tdcj + J - 1]

  INIT_FAIL(fail);
```

```
  printf("nag_opt_nlp_revcomm (e04ufc) Example Program Results\n");
  fflush(stdout);

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

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

  if (n <= 0 || nclin < 0 || ncnln < 0)
    {
      printf("At least one of n, nclin, or ncnln is invalid\n");
      exit_status = 1;
    }
  else
    {
      tda = MAX(nclin,n);
      tdcj = MAX(ncnln,n);
      tdr = n;
      nctotal = n + nclin + ncnln;
      liwork = 3*n + nclin + 2*ncnln;
      lwork = 21*n + 2;
      if (ncnln || nclin) lwork += 2*n*n + 11*nclin;
      if (ncnln) lwork += n*nclin + 2*n*ncnln + 22*ncnln - 1;


      /* Allocate memory */
      if (!(a = NAG_ALLOC(tda*MAX(1,nclin), double)) ||
          !(bl = NAG_ALLOC(nctotal, double)) ||
          !(bu = NAG_ALLOC(nctotal, double)) ||
          !(istate = NAG_ALLOC(nctotal, Integer)) ||
          !(c = NAG_ALLOC(ncnln, double)) ||
          !(cjac = NAG_ALLOC(tdcj*MAX(1,ncnln), double)) ||
          !(clamda = NAG_ALLOC(nctotal, double)) ||
          !(objgrd = NAG_ALLOC(n, double)) ||
          !(r = NAG_ALLOC(tdr*n, double)) ||
          !(x = NAG_ALLOC(n, double)) ||
          !(needc = NAG_ALLOC(ncnln, Integer)) ||
          !(iwork = NAG_ALLOC(liwork, Integer)) ||
          !(work = NAG_ALLOC(lwork, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
        }
      else
        {
          /* Read A, BL, BU and X from data file */
          if (nclin > 0)
            {
              for (i = 1; i <= nclin; ++i)
                for (j = 1; j <= n; ++j)
#ifdef _WIN32
                  scanf_s("%lf", &A(i, j));
#else
                  scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
              scanf_s("%*[^\n] ");
#else
              scanf("%*[^\n] ");
#endif
            }
```

```
          for (i = 0; i < nctotal; ++i)
#ifdef _WIN32
            scanf_s("%lf", &bl[i]);
#else
            scanf("%lf", &bl[i]);
#endif

#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
          for (i = 0; i < nctotal; ++i)
#ifdef _WIN32
            scanf_s("%lf", &bu[i]);
#else
            scanf("%lf", &bu[i]);
#endif

#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
          for (i = 0; i < n; ++i) {
#ifdef _WIN32
            scanf_s("%lf", &x[i]);
#else
            scanf("%lf", &x[i]);
#endif
          }

          /* Set all constraint Jacobian elements to zero.
             Note that this will only work when 'Derivative Level = 3'
             (the default; see Section 11.2) */

          for (j = 1; j <= n; ++j)
            for (i = 1; i <= ncnln; ++i)
              CJAC(i,j) = 0.0;

          /* Initialise nag_opt_nlp_revcomm (e04ufc) */
          nag_opt_nlp_revcomm_init("e04ufc",cwsav,5,lwsav,120,iwsav,610,
                                   rwsav,475,&fail);

          /* Set print level */
          nag_opt_nlp_revcomm_option_set_string("Print Level = 10",lwsav,iwsav,
                                                rwsav,&fail);

          /* Solve the problem */
          irevcm = 0;

          do
            {
              /* nag_opt_nlp_revcomm (e04ufc).
               * Solves the nonlinear programming (NP) problem using
               * reverse communication for evaluation of functions.
               */
              nag_opt_nlp_revcomm(&irevcm,n,nclin,ncnln,tda,tdcj,tdr,a,bl,bu,
                                  &iter,istate,c,cjac,clamda,&objf,objgrd,r,x,
                                  needc,iwork,liwork,work,lwork,cwsav,lwsav,
                                  iwsav,rwsav,&fail);

              if (irevcm == 1 || irevcm == 3)
                /* Evaluate the objective function */
                objf = x[0]*x[3]*(x[0]+x[1]+x[2]) + x[2];

              if (irevcm == 2 || irevcm == 3)
                {
                  /* Evaluate the objective gradient */
                  objgrd[0] = x[3]*(2.0*x[0]+x[1]+x[2]);
                  objgrd[1] = x[0]*x[3];
```

```
                objgrd[2] = x[0]*x[3] + 1.0;
                objgrd[3] = x[0]*(x[0]+x[1]+x[2]);
              }

            if (irevcm == 4 || irevcm == 6)
              {
                /* Evaluate the nonlinear constraint functions */
                if (needc[0] > 0)
                  c[0] = x[0]*x[0] + x[1]*x[1] + x[2]*x[2] + x[3]*x[3];
                if (needc[1] > 0)
                  c[1] = x[0]*x[1]*x[2]*x[3];
              }

            if (irevcm == 5 || irevcm == 6)
              {
                /* Evaluate the constraint Jacobian */
                if (needc[0] > 0)
                  {
                    CJAC(1,1) = 2.0*x[0];
                    CJAC(1,2) = 2.0*x[1];
                    CJAC(1,3) = 2.0*x[2];
                    CJAC(1,4) = 2.0*x[3];
                  }

                if (needc[1] > 0)
                  {
                    CJAC(2,1) = x[1]*x[2]*x[3];
                    CJAC(2,2) = x[0]*x[2]*x[3];
                    CJAC(2,3) = x[0]*x[1]*x[3];
                    CJAC(2,4) = x[0]*x[1]*x[2];
                  }
              }
          }  while (irevcm > 0);

        if (fail.code != NE_NOERROR)
          {
            printf("nag_opt_nlp_revcomm (e04ufc) failed.\n%s\n",fail.message);
            exit_status = 1;
          }
      }

    /* Deallocate any allocated arrays */
    NAG_FREE(a);
    NAG_FREE(bl);
    NAG_FREE(bu);
    NAG_FREE(istate);
    NAG_FREE(c);
    NAG_FREE(cjac);
    NAG_FREE(clamda);
    NAG_FREE(objgrd);
    NAG_FREE(r);
    NAG_FREE(x);
    NAG_FREE(needc);
    NAG_FREE(iwork);
    NAG_FREE(work);
  }
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_nlp_revcomm (e04ufc) Example Program Data
  4   1   2                              :Values of n, nclin and ncnln
  1.0   1.0   1.0   1.0                                :End of matrix a
  1.0   1.0   1.0   1.0  -1.0e+25  -1.0e+25  25.0      :End of bl
  5.0   5.0   5.0   5.0  20.0      40.0       1.0e+25  :End of bu
  1.0   5.0   5.0   1.0                                :End of x
```

10.3 Program Results

```
nag_opt_nlp_revcomm (e04ufc) Example Program Results

 *** e04ufc

 Parameters
 ----------

 Linear constraints.....         1       Variables..............         4
 Nonlinear constraints..         2

 Infinite bound size....  1.00E+20       COLD start.............
 Infinite step size.....  1.00E+20       EPS (machine precision)  1.11E-16
 Step limit.............  2.00E+00       Hessian................        NO

 Linear feasibility.....  1.05E-08       Crash tolerance........  1.00E-02
 Nonlinear feasibility..  1.05E-08       Optimality tolerance...  3.26E-12
 Line search tolerance..  9.00E-01       Function precision.....  4.37E-15

 Derivative level.......         3       Monitoring file........        -1
 Verify level...........         0

 Major iterations limit.        50       Major print level......        10
 Minor iterations limit.        50       Minor print level......         0

 Workspace provided is     IWORK(      17),  WORK(     192).
 To solve problem we need  IWORK(      17),  WORK(     192).


 Verification of the constraint gradients.
 -----------------------------------------

 The constraint Jacobian seems to be ok.

 The largest relative error was    2.29E-07  in constraint    2


 Verification of the objective gradients.
 ----------------------------------------

 The objective gradients seem to be ok.

 Directional derivative of the objective    8.15250000E-01
 Difference approximation                   8.15249734E-01


  Maj  Mnr    Step Merit Function Norm Gz  Violtn Cond Hz
    0    4 0.0E+00   1.738281E+01 7.1E-01 1.2E+01 1.0E+00
    1    1 1.0E+00   1.703169E+01 4.6E-02 1.9E+00 1.0E+00
    2    1 1.0E+00   1.701442E+01 2.1E-02 8.8E-02 1.0E+00
    3    1 1.0E+00   1.701402E+01 3.1E-04 5.4E-04 1.0E+00
    4    1 1.0E+00   1.701402E+01 7.0E-06 9.9E-08 1.0E+00
    5    1 1.0E+00   1.701402E+01 1.1E-08 4.6E-11 1.0E+00

 Exit from NP problem after     5 major iterations,
                                9 minor iterations.


 Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 V   1    LL    1.00000       1.00000       5.00000       1.088         .
 V   2    FR    4.74300       1.00000       5.00000           .      0.2570
 V   3    FR    3.82115       1.00000       5.00000           .       1.179
 V   4    FR    1.37941       1.00000       5.00000           .      0.3794


 L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 L   1    FR    10.9436          None       20.0000           .       9.056
```

```
 N Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

 N   1    UL    40.0000          None       40.0000      -0.1615     -3.5264E-11
 N   2    LL    25.0000       25.0000          None       0.5523     -2.8791E-11

 Exit e04ufc - Optimal solution found.

 Final objective value =    17.01402
```

Note: *the remainder of this document is intended for more advanced users. Section 11 contains a detailed description of the algorithm which may be needed in order to understand Sections 12 and 13. Section 12 describes the optional arguments which may be set by calls to nag_opt_nlp_revcomm_option_set_file (e04udc) and/or nag_opt_nlp_revcomm_option_set_string (e04uec). Section 13 describes the quantities which can be requested to monitor the course of the computation.*

11 Algorithmic Details

This section contains a detailed description of the method used by nag_opt_nlp_revcomm (e04ufc).

11.1 Overview

nag_opt_nlp_revcomm (e04ufc) is essentially identical to the function NPSOL described in Gill *et al.* (1986b).

At a solution of (1), some of the constraints will be *active*, i.e., satisfied exactly. An active simple bound constraint implies that the corresponding variable is *fixed* at its bound, and hence the variables are partitioned into *fixed* and *free* variables. Let C denote the m by n matrix of gradients of the active general linear and nonlinear constraints. The number of fixed variables will be denoted by n_{FX}, with n_{FR} ($n_{\mathrm{FR}} = n - n_{\mathrm{FX}}$) the number of free variables. The subscripts 'FX' and 'FR' on a vector or matrix will denote the vector or matrix composed of the elements corresponding to fixed or free variables.

A point x is a *first-order Kuhn–Tucker point* for (1) (see Powell (1974)) if the following conditions hold:

(i) x is feasible;

(ii) there exist vectors ξ and λ (*the Lagrange multiplier vectors for the bound and general constraints*) such that

$$g = C^{\mathrm{T}}\lambda + \xi \tag{2}$$

where g is the gradient of F evaluated at x and $\xi_j = 0$ if the jth variable is free.

(iii) the Lagrange multiplier corresponding to an inequality constraint active at its lower bound must be non-negative. It is non-positive for an inequality constraint active at its upper bound.

Let Z denote a matrix whose columns form a basis for the set of vectors orthogonal to the rows of C_{FR}; i.e., $C_{\mathrm{FR}}Z = 0$. An equivalent statement of the condition (2) in terms of Z is

$$Z^{\mathrm{T}}g_{\mathrm{FR}} = 0.$$

The vector $Z^{\mathrm{T}}g_{\mathrm{FR}}$ is termed the *projected gradient* of F at x. Certain additional conditions must be satisfied in order for a first-order Kuhn–Tucker point to be a solution of (1) (see Powell (1974)).

nag_opt_nlp_revcomm (e04ufc) implements a sequential quadratic programming (SQP) method. For an overview of SQP methods, see Fletcher (1987), Gill *et al.* (1981) and Powell (1983).

The basic structure of nag_opt_nlp_revcomm (e04ufc) involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that converge to x^*, a first-order Kuhn–Tucker point of (1). At a typical major iteration, the new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p \tag{3}$$

where x is the current iterate, the non-negative scalar α is the *step length,* and p is the *search direction.* (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the

iteration.) Also associated with each major iteration are estimates of the Lagrange multipliers and a prediction of the active set.

The search direction p in (3) is the solution of a quadratic programming subproblem of the form

$$\underset{p}{\text{minimize}} g^{\mathrm{T}}p + \tfrac{1}{2}p^{\mathrm{T}}Hp \quad \text{subject to} \quad \bar{l} \leq \begin{pmatrix} p \\ A_L p \\ A_N p \end{pmatrix} \leq \bar{u}, \tag{4}$$

where g is the gradient of F at x, the matrix H is a positive definite quasi-Newton approximation to the Hessian of the Lagrangian function (see Section 11.4), and A_N is the Jacobian matrix of c evaluated at x. (Finite difference estimates may be used for g and A_N; see the optional argument **Derivative Level**.) Let l in (1) be partitioned into three sections: l_B, l_L and l_N, corresponding to the bound, linear and nonlinear constraints. The vector $\bar{l}$ in (4) is similarly partitioned and is defined as

$$\bar{l}_B = l_B - x, \quad \bar{l}_L = l_L - A_L x, \quad \text{and} \quad \bar{l}_N = l_N - c,$$

where c is the vector of nonlinear constraints evaluated at x. The vector $\bar{u}$ is defined in an analogous fashion.

The estimated Lagrange multipliers at each major iteration are the Lagrange multipliers from the subproblem (4) (and similarly for the predicted active set). (The numbers of bounds, general linear and nonlinear constraints in the QP active set are the quantities `Bnd`, `Lin` and `Nln` in the monitoring file output of nag_opt_nlp_revcomm (e04ufc); see Section 13.) In nag_opt_nlp_revcomm (e04ufc), (4) is solved using an algorithm described in nag_opt_lin_lsq (e04ncc). Since solving a quadratic program is itself an iterative procedure, the *minor* iterations of nag_opt_nlp_revcomm (e04ufc) are the iterations of nag_opt_lin_lsq (e04ncc). (More details about solving the subproblem are given in Section 11.2.)

Certain matrices associated with the QP subproblem are relevant in the major iterations. Let the subscripts 'FX' and 'FR' refer to the *predicted* fixed and free variables, and let C denote the m by n matrix of gradients of the general linear and nonlinear constraints in the predicted active set. Firstly, we have available the TQ factorization of C_{FR}:

$$C_{\mathrm{FR}}Q_{\mathrm{FR}} = (0 \quad T), \tag{5}$$

where T is a nonsingular m by m reverse-triangular matrix (i.e., $t_{ij} = 0$ if $i + j < m$), and the nonsingular n_{FR} by n_{FR} matrix Q_{FR} is the product of orthogonal transformations (see Gill *et al.* (1984b)). Secondly, we have the upper triangular Cholesky factor R of the *transformed and reordered* Hessian matrix

$$R^{\mathrm{T}}R = H_Q \equiv Q^{\mathrm{T}}\tilde{H}Q, \tag{6}$$

where $\tilde{H}$ is the Hessian H with rows and columns permuted so that the free variables are first and Q is the n by n matrix

$$Q = \begin{pmatrix} Q_{\mathrm{FR}} & \\ & I_{\mathrm{FX}} \end{pmatrix} \tag{7}$$

with I_{FX} the identity matrix of order n_{FX}. If the columns of Q_{FR} are partitioned so that

$$Q_{\mathrm{FR}} = \begin{pmatrix} Z & Y \end{pmatrix},$$

then the n_Z ($n_Z \equiv n_{\mathrm{FR}} - m$) columns of Z form a basis for the null space of C_{FR}. The matrix Z is used to compute the projected gradient $Z^{\mathrm{T}}g_{\mathrm{FR}}$ at the current iterate. (The values `Nz` and `Norm Gz` printed by nag_opt_nlp_revcomm (e04ufc) give n_Z and $\|Z^{\mathrm{T}}g_{\mathrm{FR}}\|$, see Section 13.)

A theoretical characteristic of SQP methods is that the predicted active set from the QP subproblem (4) is identical to the correct active set in a neighbourhood of x^*. In nag_opt_nlp_revcomm (e04ufc), this feature is exploited by using the QP active set from the previous iteration as a prediction of the active set for the next QP subproblem, which leads in practice to optimality of the subproblems in only one iteration as the solution is approached. Separate treatment of bound and linear constraints in nag_opt_nlp_revcomm (e04ufc) also saves computation in factorizing C_{FR} and H_Q.

Once p has been computed, the major iteration proceeds by determining a step length α that produces a 'sufficient decrease' in an augmented Lagrangian *merit function* (see Section 11.3). Finally, the

approximation to the transformed Hessian matrix H_Q is updated using a modified BFGS quasi-Newton update (see Section 11.4) to incorporate new curvature information obtained in the move from x to $\bar{x}$.

On entry to nag_opt_nlp_revcomm (e04ufc), an iterative procedure from nag_opt_lin_lsq (e04ncc) is executed, starting with the user-supplied initial point, to find a point that is feasible with respect to the bounds and linear constraints (using the tolerance specified by the optional argument **Linear Feasibility Tolerance**). If no feasible point exists for the bound and linear constraints, (1) has no solution and nag_opt_nlp_revcomm (e04ufc) terminates. Otherwise, the problem functions will thereafter be evaluated only at points that are feasible with respect to the bounds and linear constraints. The only exception involves variables whose bounds differ by an amount comparable to the finite difference interval (see the discussion of the optional argument **Difference Interval**). In contrast to the bounds and linear constraints, it must be emphasized that *the nonlinear constraints will not generally be satisfied until an optimal point* is reached.

Facilities are provided to check whether the user-supplied gradients appear to be correct (see the description of the optional argument **Verify**). In general, the check is provided at the first point that is feasible with respect to the linear constraints and bounds. However, you may request that the check be performed at the initial point.

In summary, the method of nag_opt_nlp_revcomm (e04ufc) first determines a point that satisfies the bound and linear constraints. Thereafter, each iteration includes:

(a) the solution of a quadratic programming subproblem;

(b) a linesearch with an augmented Lagrangian merit function; and

(c) a quasi-Newton update of the approximate Hessian of the Lagrangian function.

These three procedures are described in more detail in Sections 11.2 to 11.4.

11.2 Solution of the Quadratic Programming Subproblem

The search direction p is obtained by solving (4) using an algorithm implemented by nag_opt_lin_lsq (e04ncc) (see Gill *et al.* (1986)), which was specifically designed to be used within an SQP algorithm for nonlinear programming.

nag_opt_lin_lsq (e04ncc) is based on a two-phase (primal) quadratic programming method. The two phases of the method are: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*) and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same functions. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function.

In general, a quadratic program must be solved by iteration. Let p denote the current estimate of the solution of (4); the new iterate $\bar{p}$ is defined by

$$\bar{p} = p + \sigma d \tag{8}$$

where, as in (3), σ is a non-negative step length and d is a search direction.

At the beginning of each iteration of nag_opt_lin_lsq (e04ncc), a *working* set is defined of constraints (general and bound) that are satisfied exactly. The vector d is then constructed so that the values of constraints in the working set remain *unaltered* for any move along d. For a bound constraint in the working set, this property is achieved by setting the corresponding element of d to zero, i.e., by fixing the variable at its bound. As before, the subscripts 'FX' and 'FR' denote selection of the elements associated with the fixed and free variables.

Let C denote the sub-matrix of rows of

$$\begin{pmatrix} A_L \\ A_N \end{pmatrix}$$

corresponding to general constraints in the working set. The general constraints in the working set will remain unaltered if

$$C_{\mathrm{FR}} d_{\mathrm{FR}} = 0, \tag{9}$$

which is equivalent to defining d_{FR} as

$$d_{\mathrm{FR}} = Z d_Z \tag{10}$$

for some vector d_Z, where Z is the matrix associated with the TQ factorization (5) of C_{FR}.

The definition of d_Z in (10) depends on whether the current p is feasible. If not, d_Z is zero except for an element γ in the jth position, where j and γ are chosen so that the sum of infeasibilities is decreasing along d. (For further details, see Gill *et al.* (1986).) In the feasible case, d_Z satisfies the equations

$$R_Z^{\mathrm{T}} R_Z d_Z = -Z^{\mathrm{T}} q_{\mathrm{FR}}, \tag{11}$$

where R_Z is the Cholesky factor of $Z^{\mathrm{T}} H_{\mathrm{FR}} Z$ and q is the gradient of the quadratic objective function ($q = g + Hp$). (The vector $Z^{\mathrm{T}} q_{\mathrm{FR}}$ is the projected gradient of the QP.) With (11), $p + d$ is the minimizer of the quadratic objective function subject to treating the constraints in the working set as equalities.

If the QP projected gradient is zero, the current point is a constrained stationary point in the subspace defined by the working set. During the feasibility phase, the projected gradient will usually be zero only at a vertex (although it may vanish at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero projected gradient implies that p minimizes the quadratic objective function when the constraints in the working set are treated as equalities. In either case, Lagrange multipliers are computed. Given a positive constant δ of the order of the ***machine precision***, the Lagrange multiplier μ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\mu_j \leq \delta$ when the jth constraint is at its *upper bound*, or if $\mu_j \geq -\delta$ when the associated constraint is at its *lower bound*. If any multiplier is nonoptimal, the current objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is nonzero, no feasible point exists. The QP algorithm will then continue iterating to determine the minimum sum of infeasibilities. At this point, the Lagrange multiplier μ_j will satisfy $-(1+\delta) \leq \mu_j \leq \delta$ for an inequality constraint at its upper bound, and $-\delta \leq \mu_j \leq (1+\delta)$ for an inequality at its lower bound. The Lagrange multiplier for an equality constraint will satisfy $\left|\mu_j\right| \leq 1 + \delta$.

The choice of step length σ in the QP iteration (8) is based on remaining feasible with respect to the satisfied constraints. During the optimality phase, if $p + d$ is feasible, σ will be taken as unity. (In this case, the projected gradient at $\bar{p}$ will be zero.) Otherwise, σ is set to σ_{M}, the step to the 'nearest' constraint, which is added to the working set at the next iteration.

Each change in the working set leads to a simple change to C_{FR}: if the status of a general constraint changes, a *row* of C_{FR} is altered; if a bound constraint enters or leaves the working set, a *column* of C_{FR} changes. Explicit representations are recurred of the matrices T, Q_{FR} and R, and of the vectors $Q^{\mathrm{T}} q$ and $Q^{\mathrm{T}} g$.

11.3 The Merit Function

After computing the search direction as described in Section 11.2, each major iteration proceeds by determining a step length α in (3) that produces a 'sufficient decrease' in the augmented Lagrangian merit function

$$L(x, \lambda, s) = F(x) - \sum_i \lambda_i (c_i(x) - s_i) + \tfrac{1}{2} \sum_i \rho_i (c_i(x) - s_i)^2, \tag{12}$$

where x, λ and s vary during the linesearch. The summation terms in (12) involve only the *nonlinear* constraints. The vector λ is an estimate of the Lagrange multipliers for the nonlinear constraints of (1). The non-negative *slack variables* $\{s_i\}$ allow nonlinear inequality constraints to be treated without introducing discontinuities. The solution of the QP subproblem (4) provides a vector triple that serves as a direction of search for the three sets of variables. The non-negative vector ρ of *penalty arguments* is initialized to zero at the beginning of the first major iteration. Thereafter, selected elements are increased whenever necessary to ensure descent for the merit function. Thus, the sequence of norms of ρ (the printed quantity `Penalty`; see Section 13) is generally nondecreasing, although each ρ_i may be reduced a limited number of times.

The merit function (12) and its global convergence properties are described in Gill *et al.* (1986a).

11.4 The Quasi-Newton Update

The matrix H in (4) is a *positive definite quasi-Newton* approximation to the Hessian of the Lagrangian function. (For a review of quasi-Newton methods, see Dennis and Schnabel (1983).) At the end of each major iteration, a new Hessian approximation $\bar{H}$ is defined as a rank-two modification of H. In nag_opt_nlp_revcomm (e04ufc), the BFGS (Broyden–Fletcher–Goldfarb–Shanno) quasi-Newton update is used:

$$\bar{H} = H - \frac{1}{s^{\mathrm{T}}Hs}Hss^{\mathrm{T}}H + \frac{1}{y^{\mathrm{T}}s}yy^{\mathrm{T}}, \tag{13}$$

where $s = \bar{x} - x$ (the change in x).

In nag_opt_nlp_revcomm (e04ufc), H is required to be positive definite. If H is positive definite, $\bar{H}$ defined by (13) will be positive definite if and only if $y^{\mathrm{T}}s$ is positive (see Dennis and Moré (1977)). Ideally, y in (13) would be taken as y_L, the change in gradient of the Lagrangian function

$$y_L = \bar{g} - \bar{A}_N^{\mathrm{T}}\mu_N - g + A_N^{\mathrm{T}}\mu_N, \tag{14}$$

where μ_N denotes the QP multipliers associated with the nonlinear constraints of the original problem. If $y_L^{\mathrm{T}}s$ is not sufficiently positive, an attempt is made to perform the update with a vector y of the form

$$y = y_L + \sum_{i=1}^{m_N}\omega_i(a_i(\hat{x})c_i(\hat{x}) - a_i(x)c_i(x)),$$

where $\omega_i \geq 0$. If no such vector can be found, the update is performed with a scaled y_L. In this case, `M` is printed to indicate that the update was modified.

Rather than modifying H itself, the Cholesky factor of the *transformed Hessian* H_Q (6) is updated, where Q is the matrix from (5) associated with the active set of the QP subproblem. The update (13) is equivalent to the following update to H_Q:

$$\bar{H}_Q = H_Q - \frac{1}{s_Q^{\mathrm{T}}H_Qs_Q}H_Qs_Qs_Q^{\mathrm{T}}H_Q + \frac{1}{y_Q^{\mathrm{T}}s_Q}y_Qy_Q^{\mathrm{T}}, \tag{15}$$

where $y_Q = Q^{\mathrm{T}}y$, and $s_Q = Q^{\mathrm{T}}s$. This update may be expressed as a *rank-one* update to R (see Dennis and Schnabel (1981)).

12 Optional Arguments

Several optional arguments in nag_opt_nlp_revcomm (e04ufc) define choices in the problem specification or the algorithm logic. In order to reduce the complexity of using nag_opt_nlp_revcomm (e04ufc) these optional arguments have associated *default values* that are appropriate for most problems. Therefore you need only specify those optional arguments 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 arguments.

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

Central Difference Interval
Cold Start
Crash Tolerance
Defaults
Derivative Level
Difference Interval
Feasibility Tolerance
Function Precision
Hessian

Infinite Bound Size
Infinite Step Size
Iteration Limit
Iters
Itns
Linear Feasibility Tolerance
Line Search Tolerance
List
Major Iteration Limit
Major Print Level
Minor Iteration Limit
Minor Print Level
Monitoring File
Nolist
Nonlinear Feasibility Tolerance
Optimality Tolerance
Print Level
Start Constraint Check At Variable
Start Objective Check At Variable
Step Limit
Stop Constraint Check At Variable
Stop Objective Check At Variable
Verify
Verify Constraint Gradients
Verify Gradients
Verify Level
Verify Objective Gradients
Warm Start

Optional arguments may be specified by calling one, or both, of nag_opt_nlp_revcomm_option_set_file (e04udc) and nag_opt_nlp_revcomm_option_set_string (e04uec) before a call to nag_opt_nlp_revcomm (e04ufc).

nag_opt_nlp_revcomm_option_set_file (e04udc) reads options from an external options file, with `Begin` and `End` as the first and last lines respectively and each intermediate line defining a single optional argument. For example,

```
Begin * Example options file
   Print level = 5
End
```

nag_opt_nlp_revcomm_option_set_file (e04udc) can then be used to read the contents of a file as opened by `nag_open_file`. nag_opt_nlp_revcomm_option_set_file (e04udc) should be consulted for a full description of this method of supplying optional arguments.

nag_opt_nlp_revcomm_option_set_string (e04uec) can be called to supply options directly, one call being necessary for each optional argument. For example,

```
e04uec("Print Level = 1",lwsav,iwsav,rwsav,&fail)
```

nag_opt_nlp_revcomm_option_set_string (e04uec) should be consulted for a full description of this method of supplying optional arguments.

All optional arguments not specified are set to their default values. Optional arguments specified are unaltered by nag_opt_nlp_revcomm (e04ufc) (unless they define invalid values) and so remain in effect for subsequent calls to nag_opt_nlp_revcomm (e04ufc).

12.1 Description of the Optional Arguments

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

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined (if no characters of an optional qualifier are underlined, the qualifier may be omitted);

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)), and ϵ_r denotes the relative precision of the objective function **Function Precision**.

Keywords and character values are case and white space insensitive.

Central Difference Interval r Default values are computed

If the algorithm switches to central differences because the forward-difference approximation is not sufficiently accurate, the value of r is used as the difference interval for every element of x. The switch to central differences is indicated by `C` at the end of each line of intermediate printout produced by the major iterations (see Section 9.1). The use of finite differences is discussed further under the optional argument **Difference Interval**.

If you supply a value for this optional parameter, a small value between 0.0 and 1.0 is appropriate.

Cold Start Default
Warm Start

This option controls the specification of the initial working set in both the procedure for finding a feasible point for the linear constraints and bounds and in the first QP subproblem thereafter. With a **Cold Start**, the first working set is chosen by nag_opt_nlp_revcomm (e04ufc) based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or 'nearly' satisfy their bounds (to within **Crash Tolerance**).

With a **Warm Start**, you must set the **istate** array and define **clamda** and **r** as discussed in Section 5. **istate** values associated with bounds and linear constraints determine the initial working set of the procedure to find a feasible point with respect to the bounds and linear constraints. **istate** values associated with nonlinear constraints determine the initial working set of the first QP subproblem after such a feasible point has been found. nag_opt_nlp_revcomm (e04ufc) will override your specification of **istate** if necessary, so that a poor choice of the working set will not cause a fatal error. For instance, any elements of **istate** which are set to -2, -1 or 4 will be reset to zero, as will any elements which are set to 3 when the corresponding elements of **bl** and **bu** are not equal. A warm start will be advantageous if a good estimate of the initial working set is available – for example, when nag_opt_nlp_revcomm (e04ufc) is called repeatedly to solve related problems.

Crash Tolerance r Default $= 0.01$

This value is used in conjunction with the optional argument **Cold Start** (the default value) when nag_opt_nlp_revcomm (e04ufc) selects an initial working set. If $0 \leq r \leq 1$, the initial working set will include (if possible) bounds or general inequality constraints that lie within r of their bounds. In particular, a constraint of the form $a_j^{\mathrm{T}}x \geq l$ will be included in the initial working set if $\left|a_j^{\mathrm{T}}x - l\right| \leq r(1 + |l|)$. If $r < 0$ or $r > 1$, the default value is used.

Defaults

This special keyword may be used to reset all optional arguments to their default values.

Derivative Level i Default $= 3$

This argument indicates which derivatives are provided during intermediate exits. The possible choices for i are the following.

i	**Meaning**
3	All elements of the objective gradient and the constraint Jacobian are provided.
2	All elements of the constraint Jacobian are provided, but some elements of the objective gradient are not specified.
1	All elements of the objective gradient are provided, but some elements of the constraint Jacobian are not specified.
0	Some elements of both the objective gradient and the constraint Jacobian are not specified.

The value $i = 3$ should be used whenever possible, since nag_opt_nlp_revcomm (e04ufc) is more reliable (and will usually be more efficient) when all derivatives are exact.

If $i = 0$ or 2, nag_opt_nlp_revcomm (e04ufc) will estimate the unspecified elements of the objective gradient, using finite differences. The computation of finite difference approximations usually increases the total run-time, since an intermediate exit to the calling program is required for each unspecified element. Furthermore, less accuracy can be attained in the solution (see Chapter 8 of Gill *et al.* (1981), for a discussion of limiting accuracy).

If $i = 0$ or 1, nag_opt_nlp_revcomm (e04ufc) will approximate unspecified elements of the constraint Jacobian. One intermediate exit is needed for each variable for which partial derivatives are not available. For example, if the Jacobian has the form

$$\begin{pmatrix} * & * & * & * \\ * & ? & ? & * \\ * & * & ? & * \\ * & * & * & * \end{pmatrix}$$

where '$*$' indicates an element provided and '?' indicates an unspecified element, nag_opt_nlp_revcomm (e04ufc) will make an intermediate exit to the calling program twice: once to estimate the missing element in column 2, and again to estimate the two missing elements in column 3. (Since columns 1 and 4 are known, they require no intermediate exits for information.)

At times, central differences are used rather than forward differences, in which case twice as many intermediate exits are needed. (The switch to central differences is not under your control.)

If $i < 0$ or $i > 3$, the default value is used.

Difference Interval r Default values are computed

This option defines an interval used to estimate derivatives by finite differences in the following circumstances:

(a) For verifying the objective and/or constraint gradients (see the description of the optional argument **Verify**).

(b) For estimating unspecified elements of the objective gradient or the constraint Jacobian.

In general, a derivative with respect to the jth variable is approximated using the interval δ_j, where $\delta_j = r\left(1 + |\hat{x}_j|\right)$, with $\hat{x}$ the first point feasible with respect to the bounds and linear constraints. If the functions are well scaled then the resulting derivative approximation should be accurate to $O(r)$. See Gill *et al.* (1981) for a discussion of the accuracy in finite difference approximations.

If a difference interval is not specified then a finite difference interval will be computed automatically for each variable by a procedure that requires up to six intermediate exits for each element. This option is recommended if the function is badly scaled or you wish to have nag_opt_nlp_revcomm (e04ufc) determine constant elements in the objective and constraint gradients.

If you supply a value for this optional parameter, a small value between 0.0 and 1.0 is appropriate.

Feasibility Tolerance r Default $= \sqrt{\epsilon}$

The scalar r defines the maximum acceptable *absolute* violations in linear and nonlinear constraints at a 'feasible' point; i.e., a constraint is considered satisfied if its violation does not exceed r. If $r < \epsilon$ or $r \geq 1$, the default value is used. Using this keyword sets both optional arguments **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance** to r, if $\epsilon \leq r < 1$. (Additional details are given under the descriptions of these optional arguments.)

Function Precision r Default $= \epsilon^{0.9}$

This argument defines ϵ_r, which is intended to be a measure of the accuracy with which the problem functions $F(x)$ and $c(x)$ can be computed. If $r < \epsilon$ or $r \geq 1$, the default value is used.

The value of ϵ_r should reflect the relative precision of $1 + |F(x)|$; i.e., ϵ_r acts as a relative precision when $|F|$ is large and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-6}. In contrast, if $F(x)$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-10}. The choice of ϵ_r can be quite complicated for badly scaled problems; see Chapter 8 of Gill *et al.* (1981) for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However, when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_r should be large enough so that nag_opt_nlp_revcomm (e04ufc) will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Hessian Default $=$ NO

This option controls the contents of the upper triangular matrix R (see Section 5). nag_opt_nlp_revcomm (e04ufc) works exclusively with the *transformed and reordered* Hessian H_Q (6), and hence extra computation is required to form the Hessian itself. If **Hessian** $=$ NO, **r** contains the Cholesky factor of the transformed and reordered Hessian. If **Hessian** $=$ YES, the Cholesky factor of the approximate Hessian itself is formed and stored in **r**. You should select **Hessian** $=$ YES if a **Warm Start** will be used for the next call to nag_opt_nlp_revcomm (e04ufc).

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

If $r > 0$, r 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 $-\infty$). If $r < 0$, the default value is used.

Infinite Step Size r Default $= \max\left(bigbnd, 10^{20}\right)$

If $r > 0$, r specifies the magnitude of the change in variables that is treated as a step to an unbounded solution. If the change in x during an iteration would exceed the value of r, the objective function is considered to be unbounded below in the feasible region. If $r \leq 0$, the default value is used.

Line Search Tolerance r Default $= 0.9$

The value r $(0 \leq r < 1)$ controls the accuracy with which the step α taken during each iteration approximates a minimum of the merit function along the search direction (the smaller the value of r, the more accurate the linesearch). The default value $r = 0.9$ requests an inaccurate search and is appropriate for most problems, particularly those with any nonlinear constraints.

If there are no nonlinear constraints, a more accurate search may be appropriate when it is desirable to reduce the number of major iterations – for example, if the objective function is cheap to evaluate, or if a substantial number of derivatives are unspecified. If $r < 0$ or $r \geq 1$, the default value is used.

Linear Feasibility Tolerance r_1 Default $= \sqrt{\epsilon}$
Nonlinear Feasibility Tolerance r_2 Default $= \epsilon^{0.33}$ or $\sqrt{\epsilon}$

The default value of r_2 is $\epsilon^{0.33}$ if **Derivative Level** $= 0$ or 1, and $\sqrt{\epsilon}$ otherwise.

The scalars r_1 and r_2 define the maximum acceptable *absolute* violations in linear and nonlinear constraints at a 'feasible' point; i.e., a linear constraint is considered satisfied if its violation does not

exceed r_1. Similarly a nonlinear constraint is considered satisfied if its violation does not exceed r_2. If $r_m < \epsilon$ or $r_m \geq 1$, the default value is used, for $m = 1, 2$.

On entry to nag_opt_nlp_revcomm (e04ufc), an iterative procedure is executed in order to find a point that satisfies the linear constraints and bounds on the variables to within the tolerance r_1. All subsequent iterates will satisfy the linear constraints to within the same tolerance (unless r_1 is comparable to the finite difference interval).

For nonlinear constraints, the feasibility tolerance r_2 defines the largest constraint violation that is acceptable at an optimal point. Since nonlinear constraints are generally not satisfied until the final iterate, the value of optional argument **Nonlinear Feasibility Tolerance** acts as a partial termination criterion for the iterative sequence generated by nag_opt_nlp_revcomm (e04ufc) (see the discussion of optional argument **Optimality Tolerance**).

These tolerances should reflect the precision of the corresponding constraints. For example, if the variables and the coefficients in the linear constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify r_1 as 10^{-6}.

List
Nolist Default

Optional argument **List** may be used to turn on printing of each optional argument specification as it is supplied. **Nolist** may then be used to suppress this printing again.

Major Iteration Limit i Default $= \max(50, 3(n + n_L) + 10n_N)$
Iteration Limit
Iters
Itns

The value of i specifies the maximum number of major iterations allowed before termination. Setting $i = 0$ and **Major Print Level** > 0 means that the workspace needed will be computed and printed, but no iterations will be performed. If $i < 0$, the default value is used.

Major Print Level i
Print Level Default $= 0$

The value of i controls the amount of printout produced by the major iterations of nag_opt_nlp_revcomm (e04ufc), as indicated below. A detailed description of the printed output is given in Section 9.1 (summary output at each major iteration and the final solution) and Section 13 (monitoring information at each major iteration). (See also the description of the optional argument **Minor Print Level**.)

The following printout is sent to `stdout`:

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output (< 80 characters; see Section 9.1) for each major iteration (no printout of the final solution).
≥ 10	The final solution and one line of summary output for each major iteration.

The following printout is sent to the `fileid` (from `nag_open_file`) given by the optional argument **Monitoring File**:

i	**Output**
< 5	No output.
≥ 5	One long line of output (> 80 characters; see Section 13) for each major iteration (no printout of the final solution).

≥ 20 At each major iteration, the objective function, the Euclidean norm of the nonlinear constraint violations, the values of the nonlinear constraints (the vector c), the values of the linear constraints (the vector $A_L x$) and the current values of the variables (the vector x).

≥ 30 At each major iteration, the diagonal elements of the matrix T associated with the TQ factorization (5) (see Section 11.1) of the QP working set and the diagonal elements of R, the triangular factor of the transformed and reordered Hessian (6) (see Section 11.1).

If **Major Print Level** ≥ 5 and the the optional argument **Monitoring File** has not been set, then the summary output for each major iteration is suppressed.

Minor Iteration Limit i Default $= \max(50, 3(n + n_L + n_N))$

The value of i specifies the maximum number of iterations for finding a feasible point with respect to the bounds and linear constraints (if any). The value of i also specifies the maximum number of minor iterations for the optimality phase of each QP subproblem. If $i \leq 0$, the default value is used.

Minor Print Level i Default $= 0$

The value of i controls the amount of printout produced by the minor iterations of nag_opt_nlp_revcomm (e04ufc) (i.e., the iterations of the quadratic programming algorithm), as indicated below. A detailed description of the printed output is given in Section 13.2. (See also the description of the optional argument **Major Print Level**.)

The following printout is sent to `stdout`:

i	**Output**
0	No output.
1	The final QP solution only.
5	One line of summary output. This consists of a subset of the information described in Section 13.2 for each minor iteration (no printout of the final QP solution).
≥ 10	The final QP solution and one line of summary output for each minor iteration.

The following printout is sent to the file referenced by the optional argument **Monitoring File**:

i	**Output**
< 5	No output.
≥ 5	One long line of output (> 80 characters; see Section 13.2) for each minor iteration (no printout of the final QP solution).
≥ 20	At each minor iteration, the current estimates of the QP multipliers, the current estimate of the QP search direction, the QP constraint values and the status of each QP constraint.
≥ 30	At each minor iteration, the diagonal elements of the matrix T associated with the TQ factorization (5) (see Section 11.1) of the QP working set and the diagonal elements of the Cholesky factor R of the transformed Hessian (6) (see Section 11.1).

If **Minor Print Level** ≥ 5 and the optional argument **Monitoring File** has not been set then the summary output for each minor iteration is suppressed.

Monitoring File i Default $= -1$

(See Section 3.2.1.1 in the Essential Introduction for further information on NAG data types.)

If **Major Print Level** ≥ 5 or **Minor Print Level** ≥ 5, monitoring information produced by nag_opt_nlp_revcomm (e04ufc) at every iteration is sent to the specified file. If **Major Print Level** < 5 and **Minor Print Level** < 5, no monitoring information is produced.

Optimality Tolerance r Default $= \epsilon_r^{0.8}$

The argument r ($\epsilon_r \leq r < 1$) specifies the accuracy to which you wish the final iterate to approximate a solution of the problem. Broadly speaking, r indicates the number of correct figures desired in the objective function at the solution. For example, if r is 10^{-6} and nag_opt_nlp_revcomm (e04ufc) terminates successfully, the final value of F should have approximately six correct figures. If $r < \epsilon_r$ or $r \geq 1$, the default value is used.

nag_opt_nlp_revcomm (e04ufc) will terminate successfully if the iterative sequence of x values is judged to have converged and the final point satisfies the first-order Kuhn–Tucker conditions (see Section 11.1). The sequence of iterates is considered to have converged at x if

$$\alpha\|p\| \leq \sqrt{r}(1 + \|x\|), \tag{16}$$

where p is the search direction and α the step length from (3). An iterate is considered to satisfy the first-order conditions for a minimum if

$$\left\|Z^{\mathrm{T}} g_{\mathrm{FR}}\right\| \leq \sqrt{r}(1 + \max(1 + |F(x)|, \|g_{\mathrm{FR}}\|)) \tag{17}$$

and

$$\left|res_j\right| \leq ftol \quad \text{for all} \quad j, \tag{18}$$

where $Z^{\mathrm{T}} g_{\mathrm{FR}}$ is the projected gradient (see Section 11.1), g_{FR} is the gradient of $F(x)$ with respect to the free variables, res_j is the violation of the jth active nonlinear constraint, and $ftol$ is the **Nonlinear Feasibility Tolerance**.

Start Objective Check At Variable	i_1	Default $= 1$
Stop Objective Check At Variable	i_2	Default $= n$
Start Constraint Check At Variable	i_3	Default $= 1$
Stop Constraint Check At Variable	i_4	Default $= n$

These keywords take effect only if **Verify Level** > 0. They may be used to control the verification of gradient elements and/or Jacobian elements computed by the calling program during intermediate exits. For example, if the first 30 elements of the objective gradient appeared to be correct in an earlier run, so that only element 31 remains questionable, it is reasonable to specify **Start Objective Check At Variable** $= 31$. If the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, the above choice would also be appropriate.

If $i_{2m-1} \leq 0$ or $i_{2m-1} > \min(n, i_{2m})$, the default value is used, for $m = 1, 2$. If $i_{2m} \leq 0$ or $i_{2m} > n$, the default value is used, for $m = 1, 2$.

Step Limit r Default $= 2.0$

If $r > 0, r$ specifies the maximum change in variables at the first step of the linesearch. In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the elements of x can lead to floating-point overflow. The argument r is therefore used to encourage evaluation of the problem functions at meaningful points. Given any major iterate x, the first point $\tilde{x}$ at which F and c are evaluated during the linesearch is restricted so that

$$\|\tilde{x} - x\|_2 \leq r\big(1 + \|x\|_2\big).$$

The linesearch may go on and evaluate F and c at points further from x if this will result in a lower value of the merit function (indicated by `L` at the end of each line of output produced by the major iterations; see Section 9.1). If `L` is printed for most of the iterations, r should be set to a larger value.

Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at wild values. The default value **Step Limit** $= 2.0$ should not affect progress on well-behaved functions, but values such as 0.1 or 0.01 may be helpful when rapidly varying functions are present. If a small value of **Step Limit** is selected then a good starting point may be required. An important application is to the class of nonlinear least squares problems. If $r \leq 0$, the default value is used.

Verify Level i Default $= 0$
Verify
Verify Constraint Gradients
Verify Gradients
Verify Objective Gradients

These keywords refer to finite difference checks on the gradient elements computed by the calling program during intermediate exits. (Unspecified gradient elements are not checked.) The possible choices for i are as follows:

i	Meaning
-1	No checks are performed.
0	Only a 'cheap' test will be performed.
≥ 1	In addition to the 'cheap' test, individual gradient elements will also be checked using a reliable (but more expensive) test.

It is possible to specify **Verify Level** $= 0$ to 3 in several ways. For example, the objective gradient will be verified if **Verify**, **Verify** = YES, **Verify Gradients**, **Verify Objective Gradients** or **Verify Level** $= 1$ is specified. The constraint gradients will be verified if **Verify** = YES or **Verify Level** $= 2$ or **Verify** is specified. Similarly, the objective and the constraint gradients will be verified if **Verify** = YES or **Verify Level** $= 3$ or **Verify** is specified.

If $0 \leq i \leq 3$, gradients will be verified at the first point that satisfies the linear constraints and bounds.

If $i = 0$, only a 'cheap' test will be performed, requiring one intermediate exit for the objective function gradients and (if appropriate) one intermediate exit for the partial derivatives of the constraints.

If $1 \leq i \leq 3$, a more reliable (but more expensive) check will be made on individual gradient elements, within the ranges specified by the **Start Objective Check At Variable** and **Stop Objective Check At Variable** keywords. A result of the form `OK` or `BAD?` is printed by nag_opt_nlp_revcomm (e04ufc) to indicate whether or not each element appears to be correct.

If $10 \leq i \leq 13$, the action is the same as for $i < 10$, except that it will take place at the user-specified initial value of x.

If $i < -1$ or $4 \leq i \leq 9$ or $i > 13$, the default value is used.

We suggest that **Verify Level** $= 3$ be used whenever a new calling program is being developed.

13 Description of Monitoring Information

13.1 Output from Major Iterations

This section describes the long line of output (> 80 characters) which forms part of the monitoring information produced by nag_opt_nlp_revcomm (e04ufc). (See also the description of the optional arguments **Major Print Level**, **Minor Print Level** and **Monitoring File**.) You can control the level of printed output (see the description of the optional argument **Major Print Level**).

When **Major Print Level** ≥ 5 and **Monitoring File** ≥ 0, the following line of output is produced at every major iteration of nag_opt_nlp_revcomm (e04ufc) and sent to the file specified by **Monitoring File**. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Maj` is the major iteration count.

`Mnr` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 11).

Note that `Mnr` may be greater than the optional argument **Minor Iteration Limit** if some iterations are required for the feasibility phase.

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Nfun` is the cumulative number of evaluations of the objective function needed for the linesearch. Evaluations needed for the estimation of the gradients by finite differences are not included. `Nfun` is printed as a guide to the amount of work required for the linesearch.

`Merit Function` is the value of the augmented Lagrangian merit function (12) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty arguments (see Section 11.3). As the solution is approached, `Merit Function` will converge to the value of the objective function at the solution.

If the QP subproblem does not have a feasible point (signified by `I` at the end of the current output line) then the merit function is a large multiple of the constraint violations, weighted by the penalty arguments. During a sequence of major iterations with infeasible subproblems, the sequence of `Merit Function` values will decrease monotonically until either a feasible subproblem is obtained or nag_opt_nlp_revcomm (e04ufc) terminates with **fail.code** = NE_NONLIN_NOT_FEASIBLE (no feasible point could be found for the nonlinear constraints).

If there are no nonlinear constraints present (i.e., **ncnln** $= 0$) then this entry contains `Objective`, the value of the objective function $F(x)$. The objective function will decrease monotonically to its optimal value when there are no nonlinear constraints.

`Norm Gz` is $\|Z^{\mathrm{T}} g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 11.2). `Norm Gz` will be approximately zero in the neighbourhood of a solution.

`Violtn` is the Euclidean norm of the residuals of constraints that are violated or in the predicted active set (not printed if **ncnln** is zero). `Violtn` will be approximately zero in the neighbourhood of a solution.

`Nz` is the number of columns of Z (see Section 11.2). The value of `Nz` is the number of variables minus the number of constraints in the predicted active set; i.e., `Nz` $= n -$ (`Bnd` $+$ `Lin` $+$ `Nln`).

`Bnd` is the number of simple bound constraints in the predicted active set.

`Lin` is the number of general linear constraints in the predicted working set.

`Nln` is the number of nonlinear constraints in the predicted active set (not printed if **ncnln** is zero).

`Penalty` is the Euclidean norm of the vector of penalty arguments used in the augmented Lagrangian merit function (not printed if **ncnln** is zero).

`Cond H` is a lower bound on the condition number of the Hessian approximation H.

`Cond Hz` is a lower bound on the condition number of the projected Hessian approximation H_Z ($H_Z = Z^{\mathrm{T}} H_{\mathrm{FR}} Z = R_Z^{\mathrm{T}} R_Z$; see (6)). The larger this number, the more difficult the problem.

`Cond T` is a lower bound on the condition number of the matrix of predicted active constraints.

`Conv` is a three-letter indication of the status of the three convergence tests (16)–(18) defined in the description of the optional argument **Optimality Tolerance**. Each letter is `T` if the test is satisfied and `F` otherwise. The three tests indicate whether:

(i) the sequence of iterates has converged;

(ii) the projected gradient (`Norm Gz`) is sufficiently small; and

(iii) the norm of the residuals of constraints in the predicted active set (`Violtn`) is small enough.

If any of these indicators is `F` when nag_opt_nlp_revcomm (e04ufc) terminates with **fail.code** = NE_NOERROR, you should check the solution carefully.

`M` is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive definite (see Section 11.4).

`I` is printed if the QP subproblem has no feasible point.

`C` is printed if central differences have been used to compute the unspecified objective and constraint gradients. If the value of `Step` is zero then the switch to central differences was made because no lower point could be found in the linesearch. (In this case, the QP subproblem is resolved with the central difference gradient and Jacobian.) If the value of `Step` is nonzero then central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 11.1).

`L` is printed if the linesearch has produced a relative change in x greater than the value defined by the optional argument **Step Limit**. If this output occurs frequently during later iterations of the run, optional argument **Step Limit** should be set to a larger value.

On entry: need not be initialized if the (default) optional argument **Cold Start** is used.

If the optional argument **Warm Start** has been chosen, **r** must contain the upper triangular Cholesky factor R of the initial approximation of the Hessian of the Lagrangian function, with the variables in the natural order. Elements not in the upper triangular part of **r** are assumed to be zero and need not be assigned.

On exit: if **Hessian** = NO, **r** contains the upper triangular Cholesky factor R of $Q^{\mathrm{T}}\tilde{H}Q$, an estimate of the transformed and reordered Hessian of the Lagrangian at x (see (6) in Section 11.1). If **Hessian** = YES, **r** contains the upper triangular Cholesky factor R of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

13.2 Output from Minor Iterations

This section describes the long line of output (> 80 characters) which forms part of the monitoring information for minor iterations produced by nag_opt_nlp_revcomm (e04ufc). (See also the description of the optional arguments **Monitoring File** and **Minor Print Level**.) You can control the level of printed output.

To aid interpretation of the printed results, the following convention is used for numbering the constraints: indices 1 through n refer to the bounds on the variables, and indices $n+1$ through $n+n_L$ refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

When **Minor Print Level** ≥ 5 and **Monitoring File** ≥ 0, the following line of output is produced at every iteration and sent to the file specified by optional argument **Monitoring File**. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Itn` is the iteration count.

`Jdel` is the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` is the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` is the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. During the optimality phase, the step can be greater than one only if the factor R_Z is singular.

`Ninf` is the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Objective` is the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If x is feasible, `Objective` is the value of the objective function of (1). The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase the value of the objective function will be nonincreasing. During the feasibility phase the number of constraint infeasibilities will not increase until either a feasible point is found or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Bnd` is the number of simple bound constraints in the current working set.

`Lin` is the number of general linear constraints in the current working set.

`Art` is the number of artificial constraints in the working set.

`Zr` `Zr` is the dimension of the subspace in which the objective function is currently being minimized. The value of `Zr` is the number of variables minus the number of constraints in the working set; i.e., $\texttt{Zr} = n - (\texttt{Bnd} + \texttt{Lin} + \texttt{Art})$.

The value of n_Z, the number of columns of Z can be calculated as $n_Z = n - (\texttt{Bnd} + \texttt{Lin})$. A zero value of n_Z implies that x lies at a vertex of the feasible region.

`Norm Gz` is $\left\|Z_1^{\mathrm{T}} g_{\mathrm{FR}}\right\|$, the Euclidean norm of the reduced gradient with respect to Z_1. During the optimality phase, this norm will be approximately zero after a unit step.

`Norm Gf` is the Euclidean norm of the gradient function with respect to the free variables, i.e., variables not currently held at a bound.

`Cond T` is a lower bound on the condition number of the working set.

`Cond Rz` is a lower bound on the condition number of the triangular factor R_1 (the first `Zr` rows and columns of the factor R_Z). If the problem is specified to be of type LP or the estimated rank of the data matrix A is zero then `Cond Rz` is not printed.