

NAG C Library Function Document

nag_opt_sparse_nlp_solve (e04vhc)

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 9 of this document. Refer to the additional Sections 10, 11 and 12 for a detailed description of the algorithm, the specification of the optional arguments and a description of the monitoring information produced by the function.

1 Purpose

nag_opt_sparse_nlp_solve (e04vhc) solves sparse nonlinear programming problems.

2 Specification

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

void nag_opt_sparse_nlp_solve (Nag_Start start, Integer nf, Integer n,
    Integer nxname, Integer nfname, double objadd, Integer objrow,
    const char *prob,
    void (*usrfun)(Integer *status, Integer n, const double x[], Integer needf,
        Integer nf, double f[], Integer needg, Integer leng, double g[],
        Nag_Comm *comm),
    const Integer iafun[], const Integer javar[], const double a[], Integer lena,
    Integer nea, const Integer igfun[], const Integer igvar[], Integer leng,
    Integer neg, const double xlow[], const double xupp[], const char *xnames[],
    const double flow[], const double fupp[], const char *fnames[], double x[],
    Integer xstate[], double xmul[], double f[], Integer fstate[], double fmul[],
    Integer *ns, Integer *ninf, double *sinf, Nag_E04State *state, Nag_Comm *comm,
    NagError *fail)
```

Before calling nag_opt_sparse_nlp_solve (e04vhc), or one of the option setting functions nag_opt_sparse_nlp_option_set_file (e04vkc), nag_opt_sparse_nlp_option_set_string (e04vlc), nag_opt_sparse_nlp_option_set_integer (e04vmc) or nag_opt_sparse_nlp_option_set_double (e04vnc), function nag_opt_sparse_nlp_init (e04vgc) **must** be called. The specification for nag_opt_sparse_nlp_init (e04vgc) is:

```
void nag_opt_sparse_nlp_init (Nag_E04State *state, NagError *fail)
```

The contents of **state** **must not** be altered between calling functions nag_opt_sparse_nlp_init (e04vgc), nag_opt_sparse_nlp_solve (e04vhc), nag_opt_sparse_nlp_jacobian (e04vjc), nag_opt_sparse_nlp_option_set_file (e04vkc), nag_opt_sparse_nlp_option_set_string (e04vlc), nag_opt_sparse_nlp_option_set_integer (e04vmc) and nag_opt_sparse_nlp_option_set_double (e04vnc).

3 Description

nag_opt_sparse_nlp_solve (e04vhc) is designed to minimize a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs of the form

$$\underset{x}{\text{minimize}} f_0(x) \quad \text{subject to } l \leq \begin{pmatrix} x \\ f(x) \\ A_L x \end{pmatrix} \leq u, \quad (1)$$

where l and u are constant lower and upper bounds, $f_0(x)$ is a smooth scalar objective function, A_L is a

sparse matrix, and $f(x)$ is a vector of smooth nonlinear constraint functions $\{f_i(x)\}$. An optional argument **Maximize** (see Section 11.2) may specify that $\{f_0(x)\}$ should be maximized instead of minimized.

Ideally, the first derivatives (gradients) of $f_0(x)$ and $f_i(x)$ should be known and coded by you. If only some of the gradients are known, `nag_opt_sparse_nlp_solve` (e04vhc) will estimate the missing ones with finite differences.

If $f_0(x)$ is linear and $f(x)$ is absent, (1) is a linear program (LP) and `nag_opt_sparse_nlp_solve` (e04vhc) applies the primal simplex method (see Dantzig (1963)). Sparse basis factors are maintained by LUSOL (see Gill *et al.* (1987)) as in MINOS (see Murtagh and Saunders (1995)).

If only the objective is nonlinear, the problem is linearly constrained (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). For both cases `nag_opt_sparse_nlp_solve` (e04vhc) applies a sparse sequential quadratic programming (SQP) method (see Gill *et al.* (2002)), using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver `nag_opt_nlp_solve` (e04wdc) (see Gill *et al.* (1986c) and Gill *et al.* (1992)).

It is suitable for nonlinear problems with thousands of constraints and variables, and is efficient if many constraints and bounds are active at a solution. (Thus, ideally there should not be thousands of degrees of freedom.)

`nag_opt_sparse_nlp_solve` (e04vhc) allows linear and nonlinear constraints and variables to be entered in an *arbitrary order*, and uses one user-supplied function to define all the nonlinear functions.

The optimization problem is assumed to be in the form

$$\underset{x}{\text{minimize}} F_{\text{obj}}(x) \quad \text{subject to} \quad l_x \leq x \leq u_x, \quad l_F \leq F(x) \leq u_F, \quad (2)$$

where the upper and lower bounds are constant, $F(x)$ is a vector of smooth linear and nonlinear constraint functions $\{F_i(x)\}$, and $F_{\text{obj}}(x)$ is one of the components of F to be minimized, as specified by the input argument **objrow**. (The option **Maximize** specifies that $F_{\text{obj}}(x)$ should be maximized instead of minimized.) `nag_opt_sparse_nlp_solve` (e04vhc) reorders the variables and constraints so that the problem is in the form (1).

Ideally, the first derivatives (gradients) of F_i should be known and coded by you. If only some gradients are known, `nag_opt_sparse_nlp_solve` (e04vhc) estimates the missing ones with finite differences.

Note that upper and lower bounds are specified for all variables and functions. This form allows full generality in specifying various types of constraint. Special values are used to indicate absent bounds ($l_j = -\infty$ or $u_j = +\infty$ for appropriate j). Free variables and free constraints ('free rows') are ones that have both bounds infinite. Fixed variables and equality constraints have $l_j = u_j$.

In general, the components of F are *structured* in the sense that they are formed from sums of linear and nonlinear functions of just some of the variables. This structure can be exploited by `nag_opt_sparse_nlp_solve` (e04vhc).

In many cases, the vector $F(x)$ is a sum of linear and nonlinear functions. `nag_opt_sparse_nlp_solve` (e04vhc) allows these terms to be specified separately, so that the linear part is defined just once by the input arguments **iafun**, **javar** and **a**. Only the nonlinear part is recomputed at each x .

Suppose that each component of $F(x)$ is of the form

$$F_i(x) = f_i(x) + \sum_{j=1}^n A_{ij}x_j,$$

where $f_i(x)$ is a nonlinear function (possibly zero) and the elements A_{ij} are constant. The $n \times n$ Jacobian of $F(x)$ is the sum of two sparse matrices of the same size $F'(x) = G(x) + A$, where $G(x) = f'(x)$ and A is the matrix with elements $\{A_{ij}\}$. The two matrices must be *non-overlapping* in the sense that each element of the Jacobian $F'(x) = G(x) + A$ is an element of $G(x)$ or an element of A , but *not both*. The element cannot be split between $G(x)$ and A .

For example, the function

$$F(x) = \begin{pmatrix} 3x_1 + e^{x_2}x_4 + x_2^2 + 4x_4 - x_3 + x_5 \\ x_2 + x_3^2 + \sin x_4 - 3x_5 \\ x_1 - x_3 \end{pmatrix}$$

can be written as

$$F(x) = f(x) + Ax = \begin{pmatrix} e^{x_2}x_4 + x_2^2 + 4x_4 \\ x_3^2 + \sin x_4 \\ 0 \end{pmatrix} + \begin{pmatrix} 3x_1 - x_3 + x_5 \\ x_2 - 3x_5 \\ x_1 - x_3 \end{pmatrix},$$

in which case

$$F'(x) = \begin{pmatrix} 3 & e^{x_2}x_4 + 2x_2 & -1 & e^{x_2} + 4 & 1 \\ 0 & 1 & 2x_3 & \cos x_4 & -3 \\ 1 & 0 & -1 & 0 & 0 \end{pmatrix}$$

can be written as $F'(x) = f'(x) + A = G(x) + A$, where

$$G(x) = \begin{pmatrix} 0 & e^{x_2}x_4 + 2x_2 & 0 & e^{x_2} + 4 & 0 \\ 0 & 0 & 2x_3 & \cos x_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & -3 \\ 1 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

Note: that element $e^{x_2} + 4$ of $F'(x)$ appears in $G(x)$ and is not split between $G(x)$ and A although it contains a linear term.

The non-zero elements of A and G are provided to `nag_opt_sparse_nlp_solve` (e04vhc) in co-ordinate form. The elements of A are entered as triples (i, j, A_{ij}) in the arrays **iafun**, **javar** and **a**. The sparsity pattern G is entered as pairs (i, j) in the arrays **igfun** and **igvar**. The corresponding entries G_{ij} (any that are known) are assigned to appropriate array elements **g(k)** in the user-supplied function **usrfun**.

The elements of A and G may be stored in any order. Duplicate entries are ignored. **igfun** and **igvar** may be defined automatically by function `nag_opt_sparse_nlp_jacobian` (e04vjc) when **Derivative Option** = 0 (see Section 11.2) is specified and **usrfun** does not provide any gradients.

Throughout this document the symbol ϵ is used to represent the *machine precision* (see `nag_machine_precision` (X02AJC)).

`nag_opt_sparse_nlp_solve` (e04vhc) is based on SNOPTA, which is part of the SNOPT package described in Gill *et al.* (2004).

4 References

Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press

Eldersveld S K (1991) Large-scale sequential quadratic programming algorithms *PhD Thesis* Department of Operations Research, Stanford University, Stanford

Fourer R (1982) Solving staircase linear programs by the simplex method *Math. Programming* **23** 274–313

Gill P E, Murray W and Saunders M A (1999) Users' guide for SQOPT 5.3: a Fortran package for large-scale linear and quadratic programming *Report SOL 99* Department of Operations Research, Stanford University

Gill P E, Murray W and Saunders M A (2002) *SNOPT: An SQP Algorithm for Large-scale Constrained Optimization* **12** 979–1006 SIAM J. Optim.

Gill P E, Murray W and Saunders M A (2004) User's Guide for SNOPT Version 7.1, A Fortran Package for Large-Scale Nonlinear Programming *Report NA 04-1* Department of Mathematics, University of California, San Diego URL: <ftp://www.scicomp.ucsd.edu/pub/peg/reports/snopt7.pdf>

Gill P E, Murray W, Saunders M A and Wright M H (1986c) 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, Saunders M A and Wright M H (1987) Maintaining *LU* factors of a general sparse matrix *Linear Algebra and its Applics.* **88/89** 239–270

Gill P E, Murray W, Saunders M A and Wright M H (1992) Some theoretical properties of an augmented Lagrangian merit function *Advances in Optimization and Parallel Computing* (ed P M Pardalos) 101–128 North Holland

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 (1978) Large-scale linearly constrained optimization *14* 41–72 Math. Program.

Murtagh B A and Saunders M A (1982) A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints *Math. Program. Stud.* **16** 84–118

Murtagh B A and Saunders M A (1995) MINOS 5.4 Users' Guide *Report SOL 83-20R* Department of Operations Research, Stanford University

5 Arguments

1: **start** – Nag_Start *Input*

On entry: indicates how a starting basis (and certain other items) are to be obtained.

start = Nag_Cold (Cold Start)

Requests that the Crash procedure be used to choose an initial basis, unless a basis file is provided via option **Old Basis File**, **Insert File** or **Load File** (see Section 11.2).

start = Nag_BasisFile

Is the same as **start = Nag_Cold** but is more meaningful when a basis file is given.

start = Nag_Warm (Warm Start)

Means that a basis is already defined in **xstate** and **fstate** (probably from an earlier call).

Constraint: **start = Nag_Cold**, **Nag_BasisFile** or **Nag_Warm**.

2: **nf** – Integer *Input*

On entry: *nf*, the number of problem functions in $F(x)$, including the objective function (if any) and the linear and nonlinear constraints. Simple upper and lower bounds on x can be defined using the arguments **xlow** and **xupp** defined below and should not be included in F .

Constraint: **nf** > 0.

3: **n** – Integer *Input*

On entry: *n*, the number of variables.

Constraint: **n** > 0.

4: **nxname** – Integer *Input*

On entry: the number of names provided in the array **xnames**.

nxname = 1

There are no names provided and generic names will be used in the output.

nxname = n

Names for all variables must be provided and will be used in the output.

Constraint: **nxname = 1** or **n**.

- 5: **nfname** – Integer *Input*
On entry: the number of names provided in the array **fnames**.
nfname = 1
 There are no names provided and generic names will be used in the output.
nfname = **nf**
 Names for all functions must be provided and will be used in the output.
Constraint: **nfname** = 1 or **nf**.
- 6: **objadd** – double *Input*
On entry: is a constant that will be added to the objective row $f_0(\mathbf{objrow})$ for printing purposes. Typically, **objadd** = 0.0.
- 7: **objrow** – Integer *Input*
On entry: says which row of F is to act as the objective function. If there is no such vector, **objrow** = 0 and `nag_opt_sparse_nlp_solve` (e04vhc) will attempt to find a point such that $l_F \leq F(x) \leq u_F$ and $l_x \leq x \leq u_x$.
Constraint: $1 \leq \mathbf{objrow} \leq \mathbf{nf}$ or **objrow** = 0 (or a feasible point problem).
- 8: **prob** – const char * *Input*
On entry: the name for the problem. **prob** is used in the printed solution and in some functions that output basis files. Only the first eight characters of **prob** are significant.
- 9: **usrfun** – function, supplied by the user *External Function*
usrfun must define the nonlinear portion $f(x)$ of the problem functions $F(x) = f(x) + Ax$, along with its gradient elements $G_{ij}(x) = \frac{\partial f_i(x)}{\partial x_j}$. A dummy function is needed even if $f(x) = 0$ and all functions are linear.
 In general, **usrfun** should return all function and gradient values on every entry except perhaps the last. This provides maximum reliability and corresponds to the default option setting, **Derivative Option** = 1 (see Section 11.2).
 The elements of $G(x)$ are stored in the array $\mathbf{g}[i - 1]$, for $i = 1, 2, \dots, \mathbf{leng}$, in the order specified by the input arrays **igfun** and **jgvar**.
 In practice it is often convenient *not* to code gradients. `nag_opt_sparse_nlp_solve` (e04vhc) is able to estimate them by finite differences, using a call to **usrfun** for each variable x_j for which some $\frac{\partial f_i(x)}{\partial x_j}$ needs to be estimated. However, this reduces the reliability of the optimization algorithm, and it can be very expensive if there are many such variables x_j .
 As a compromise, `nag_opt_sparse_nlp_solve` (e04vhc) allows you to code *as many gradients as you like*. This option is implemented as follows. Just before **usrfun** is called, each element of the derivative array **g** is initialized to a specific value. On exit, any element retaining that value must be estimated by finite differences.
 Some rules of thumb follow:
- (i) for maximum reliability, compute all gradients;
 - (ii) if the gradients are expensive to compute, specify **Nonderivative LineSearch** and use the value of the input argument **needg** to avoid computing them on certain entries. (There is no need to compute gradients if **needg** = 0 on entry to **usrfun**.);
 - (iii) if not all gradients are known, you must specify **Derivative Option** = 0. You should still compute as many gradients as you can. (It often happens that some of them are constant or zero.);

- (iv) again, if the known gradients are expensive, don't compute them if **needg** = 0 on entry to **usrfun**;
- (v) use the input argument **status** to test for special actions on the first or last entries.
- (vi) while **usrfun** is being developed, use the **Verify Level** option to check the computation of gradients that are supposedly known.
- (vii) **usrfun** is not called until the linear constraints and bounds on x are satisfied. This helps confine x to regions where the functions $f_i(x)$ are likely to be defined. However, be aware of the **Minor Feasibility Tolerance** if the functions have singularities on the constraint boundaries.
- (viii) set **status** = -1 if some of the functions are undefined. The line search will shorten the step and try again.
- (ix) set **status** ≤ -2 if you want nag_opt_sparse_nlp_solve (e04vhc) to stop.

Its specification is:

```
void usrfun (Integer *status, Integer n, const double x[], Integer needf,
            Integer nf, double f[], Integer needg, Integer leng, double g[],
            Nag_Comm *comm)
```

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

On entry: indicates the first and last calls to **usrfun**.

status = 0

There is nothing special about the current call to **usrfun**.

status = 1

nag_opt_sparse_nlp_solve (e04vhc) is calling your function for the *first* time. You may wish to do something special such as read data from a file.

status ≥ 2

nag_opt_sparse_nlp_solve (e04vhc) is calling your function for the *last* time. You may wish to perform some additional computation on the final solution. In particular,

- if **status** = 2, the current x is *optimal*;
- if **status** = 3, the problem appears to be infeasible;
- if **status** = 4, the problem appears to be unbounded;
- if **status** = 5, an iterations limit was reached.

If the functions are expensive to evaluate, it may be desirable to do nothing on the last call. The first executable statement could be

```
if (*status ≥ 2) return;
```

On exit: may be used to indicate that you are unable to evaluate f or its gradients at the current x . (For example, the problem functions may not be defined there).

During the line search, $f(x)$ is evaluated at points $x = x_k + \alpha p_k$ for various steplengths α , where $f(x_k)$ has already been evaluated satisfactorily. For any such x , if you set **status** = -1, nag_opt_sparse_nlp_solve (e04vhc) will reduce α and evaluate f again (closer to x_k , where $f(x_k)$ is more likely to be defined).

If for some reason you wish to terminate the current problem, set **status** ≤ -2.

2: **n** – Integer *Input*

On entry: n , the number of variables, as defined in the call to nag_opt_sparse_nlp_solve (e04vhc).

3:	x[n] – const double	<i>Input</i>
	<i>On entry:</i> the variables x at which the problem functions are to be calculated. The array x must not be altered.	
4:	needf – Integer	<i>Input</i>
5:	nf – Integer	<i>Input</i>
6:	f[nf] – double	<i>Input/Output</i>
	<i>On entry:</i> concerns the calculation of $f(x)$.	
	nf is the length of the full vector $F(x) = f(x) + Ax$ as defined in the call to <code>nag_opt_sparse_nlp_solve</code> (e04vhc).	
	needf indicates if f must be assigned during this call of usrfun .	
	needf = 0	
	f is not required and is ignored.	
	needf > 0	
	The components of $f(x)$ corresponding to the nonlinear part of $F(x)$ must be calculated and assigned to f .	
	If $F_i(x)$ is linear and completely defined by the i th row of A , A'_i , then the associated value of $f(x)$ is ignored and need not be assigned. However, if $F_i(x)$ has a nonlinear portion $f_i(x)$ that happens to be zero at x , then it is still necessary to set $f_i(x) = 0$. If the linear part A'_i of a nonlinear $F_i(x)$ is provided using the arrays iafun , javar and a , then it must not be computed again as part of $f_i(x)$.	
	To simplify the code, you may ignore the value of needf and compute $f(x)$ on every entry to usrfun .	
	needf may also be ignored with Derivative Linesearch and Derivative Option = 1 (see Section 11.2). In this case, needf is always 1, and f must always be assigned.	
	<i>On exit:</i> f contains the computed functions $f(x)$ (except perhaps if needf = 0).	
7:	needg – Integer	<i>Input</i>
8:	leng – Integer	<i>Input</i>
9:	g[leng] – double	<i>Input/Output</i>
	<i>On entry:</i> concerns the calculations of the derivatives of the function $f(x)$.	
	leng is the length of the co-ordinate arrays jgvar and igfun in the call to <code>nag_opt_sparse_nlp_solve</code> (e04vhc).	
	needg indicates if g must be assigned during this call of usrfun .	
	needg = 0	
	g is not required and is ignored.	
	needg > 0	
	The partial derivatives of $f(x)$ must be calculated and assigned to g . The value of $\mathbf{g}[k - 1]$ should be $\frac{\partial f_i(x)}{\partial x_j}$, where $i = \mathbf{igfun}[k - 1]$, $j = \mathbf{jgvar}[k - 1]$, for $k = 1, 2, \dots, \mathbf{leng}$.	
	<i>On exit:</i> g contains the computed derivatives $G(x)$ (except perhaps if needg = 0).	
	These derivative elements must be stored in g in exactly the same positions as implied by the definitions of arrays igfun and jgvar . There is no internal check for consistency (except indirectly via the Verify Level option), so great care is essential.	

10: **comm** – Nag_Comm * *Communication Structure*
 Pointer to structure of type **Nag_Comm**; the following members are relevant to **usrfun**.
user – double *
iuser – Integer *
p – Pointer

The type Pointer will be void *. Before calling nag_opt_sparse_nlp_solve (e04vhc) these pointers may be allocated memory by the user and initialized with various quantities for use by **usrfun** when called from nag_opt_sparse_nlp_solve (e04vhc).

10: **iafun**[**lena**] – const Integer *Input*
 11: **javar**[**lena**] – const Integer *Input*
 12: **a**[**lena**] – const double *Input*

On entry: define the co-ordinates (i, j) and values A_{ij} of the non-zero elements of the linear part A of the function $f_0(x) = f(x) + Ax$.

In particular, the **nea** triples $(\mathbf{iafun}[k-1], \mathbf{javar}[k-1], \mathbf{a}[k-1])$ define the row and column indices $i = \mathbf{iafun}[k-1]$ and $j = \mathbf{javar}[k-1]$ of the element $A_{ij} = \mathbf{a}[k-1]$.

The co-ordinates may define the elements of A in any order.

13: **lena** – Integer *Input*

On entry: the dimension of the arrays **iafun**, **javar** and **a** that holds (i, j, A_{ij}) as declared in the function from which nag_opt_sparse_nlp_solve (e04vhc) is called.

Constraint: **lena** ≥ 1 .

14: **nea** – Integer *Input*

On entry: is the number of non-zero entries in A such that $F(x) = f(x) + Ax$.

Constraint: $0 \leq \mathbf{nea} \leq \mathbf{lena}$.

15: **igfun**[**leng**] – const Integer *Input*

16: **jgvar**[**leng**] – const Integer *Input*

On entry: define the co-ordinates (i, j) of the non-zero elements of G , the nonlinear part of the derivatives $J(x) = G(x) + A$ of the function $F(x) = f(x) + Ax$. nag_opt_sparse_nlp_jacobian (e04vjc) may be used to define these two arrays.

The co-ordinates can define the elements of G in any order. However, **usrfun** must define the actual elements of **g** in exactly the same order as defined by the co-ordinates (**igfun**, **jgvar**).

17: **leng** – Integer *Input*

On entry: the dimension of the arrays **igfun** and **jgvar** that define the varying Jacobian elements (i, j, G_{ij}) as declared in the function from which nag_opt_sparse_nlp_solve (e04vhc) is called.

Constraint: **leng** ≥ 1 .

18: **neg** – Integer *Input*

On entry: the number of non-zero entries in G .

Constraint: $0 \leq \mathbf{neg} \leq \mathbf{leng}$.

19: **xlow**[**n**] – const double *Input*

20: **xupp**[**n**] – const double *Input*

On entry: contain the lower and upper bounds l_x and u_x on the variables x .

To specify a non-existent lower bound $[l_x]_j = -\infty$, set $\mathbf{xlow}[j - 1] \leq -bigbnd$, where *bigbnd* is the **Infinite Bound Size**. To specify a non-existent upper bound $\mathbf{xupp}[j - 1] \geq bigbnd$.

To fix the *j*th variable (say, $x_j = \beta$, where $|\beta| < bigbnd$), set $\mathbf{xlow}[j - 1] = \mathbf{xupp}[j - 1] = \beta$.

Constraint: $\mathbf{xlow}[i - 1] \leq \mathbf{xupp}[i - 1]$, for $i = 1, 2, \dots, \mathbf{n}$.

21: $\mathbf{xnames}[\mathbf{nxname}]$ – const char * *Input*

On entry: the optional names for the variables.

If $\mathbf{nxname} = 1$, \mathbf{xnames} is not referenced and default names will be used for output.

If $\mathbf{nxname} = \mathbf{n}$, $\mathbf{xnames}[j - 1]$ must contain the name of the *j*th variable, for $j = 1, \dots, \mathbf{nxname}$.

Note: that only the first eight characters of the rows of \mathbf{xnames} are significant.

22: $\mathbf{flow}[\mathbf{nf}]$ – const double *Input*

23: $\mathbf{fupp}[\mathbf{nf}]$ – const double *Input*

On entry: contain the lower and upper bounds l_F and u_F on $F(x)$.

To specify a non-existent lower bound $[l_F]_i = -\infty$, set $\mathbf{flow}[i - 1] \leq -bigbnd$. For a non-existent upper bound $[u_F]_i = \infty$, set $\mathbf{fupp}[i - 1] \geq bigbnd$.

To make the *i*th constraint an *equality* constraint (say, $F_i = \beta$, where $|\beta| < bigbnd$), set $\mathbf{flow}[i - 1] = \mathbf{fupp}[i - 1] = \beta$.

Constraint: $\mathbf{flow}[i - 1] \leq \mathbf{fupp}[i - 1]$, for $i = 1, 2, \dots, \mathbf{n}$.

24: $\mathbf{fnames}[\mathbf{nfname}]$ – const char * *Input*

On entry: the optional names for the problem functions.

If $\mathbf{nfname} = 1$, \mathbf{fnames} is not referenced and default names will be used for the output.

If $\mathbf{nfname} = \mathbf{nf}$, $\mathbf{fnames}[i - 1]$ should contain the name of the *i*th row of F , for $i = 1, \dots, \mathbf{nfname}$.

Note: that only the first eight characters of the rows of \mathbf{fnames} are significant.

25: $\mathbf{x}[\mathbf{n}]$ – double *Input/Output*

On entry: an initial estimate of the variables x . See the following description of \mathbf{xstate} .

On exit: the final values of the variable x .

26: $\mathbf{xstate}[\mathbf{n}]$ – Integer *Input/Output*

On entry: the initial state for each variable x .

If $\mathbf{start} = \mathbf{Nag_Cold}$ or $\mathbf{Nag_BasisFile}$ and no basis information is provided (the optional arguments **Old Basis File**, **Insert File** and **Load File** are all set to 0; the default, see Section 11.2) \mathbf{x} and \mathbf{xstate} must be defined.

If nothing special is known about the problem, or if there is no wish to provide special information, you may set $\mathbf{x}[j - 1] = 0.0$, $\mathbf{xstate}[j - 1] = 0$, for all $j = 1, \dots, \mathbf{n}$. If you set $\mathbf{x}[j - 1] = \mathbf{xlow}[j - 1]$ set $\mathbf{xstate}[j - 1] = 4$; if you set $\mathbf{x}[j - 1] = \mathbf{xupp}[j - 1]$ then set $\mathbf{xstate}[j - 1] = 5$. In this case a Crash procedure is used to select an initial basis.

If $\mathbf{start} = \mathbf{Nag_Cold}$ or $\mathbf{Nag_BasisFile}$ and basis information is provided (at least one of the optional arguments **Old Basis File**, **Insert File** and **Load File** is non-zero, see Section 11.2) \mathbf{x} and \mathbf{xstate} need not be set.

If $\mathbf{start} = \mathbf{Nag_Warm}$, \mathbf{x} and \mathbf{xstate} must be set (probably from a previous call). In this case $\mathbf{xstate}[j - 1]$ must be 0, 1, 2 or 3, for $j = 1, \dots, \mathbf{n}$.

On exit: the final state of the variables.

$\mathbf{xstate}[j - 1]$	State of variable j	Usual value of $\mathbf{x}[j - 1]$
0	nonbasic	$\mathbf{xlow}[j - 1]$
1	nonbasic	$\mathbf{xupp}[j - 1]$
2	superbasic	Between $\mathbf{xlow}[j - 1]$ and $\mathbf{xupp}[j - 1]$
3	basic	Between $\mathbf{xlow}[j - 1]$ and $\mathbf{xupp}[j - 1]$

Basic and superbasic variables may be outside their bounds by as much as the **Minor Feasibility Tolerance**. Note that if scaling is specified, the feasibility tolerance applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the value of Primal infeasibility output to **Print File**.

Very occasionally some nonbasic variables may be outside their bounds by as much as the **Minor Feasibility Tolerance**, and there may be some nonbasics for which $\mathbf{x}[j - 1]$ lies strictly between its bounds.

If $\mathbf{ninf} > 0$, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by \mathbf{sinf} if scaling was not used).

Constraint: $0 \leq \mathbf{xstate}[j - 1] \leq 5$, for $j = 1, \dots, \mathbf{n}$.

- 27: $\mathbf{xmul}[\mathbf{n}]$ – double *Output*
On exit: the vector of the dual variables (Lagrange-multipliers) for the simple bounds $l_x \leq x \leq u_x$.
- 28: $\mathbf{f}[\mathbf{nf}]$ – double *Input/Output*
On entry: an initial value for the problem functions F . See the following description of \mathbf{fstate} .
On exit: the final values for the problem functions F (the values F at the final point \mathbf{x}).
- 29: $\mathbf{fstate}[\mathbf{nf}]$ – Integer *Input/Output*
On entry: the initial state for the problem functions F .

If $\mathbf{start} = \mathbf{Nag_Cold}$ or $\mathbf{Nag_BasisFile}$ and no basis information is provided (the optional arguments **Old Basis File**, **Insert File** and **Load File** are all set to 0; the default, see Section 11.2), \mathbf{f} and \mathbf{fstate} must be defined.

If nothing special is known about the problem, or if there is no wish to provide special information, you may set $\mathbf{f}[i - 1] = 0.0$, $\mathbf{fstate}[i - 1] = 0$, for all $i = 1, \dots, \mathbf{nf}$. Less trivially, to say that the optimal value of function $\mathbf{f}[i - 1]$ will probably be equal to one of its bounds, set $\mathbf{f}[i - 1] = \mathbf{flow}[i - 1]$ and $\mathbf{fstate}[i - 1] = 4$ or $\mathbf{f}[i - 1] = \mathbf{fupp}[i - 1]$ and $\mathbf{fstate}[i - 1] = 5$ as appropriate. In this case a Crash procedure is used to select an initial basis.

If $\mathbf{start} = \mathbf{Nag_Cold}$ or $\mathbf{Nag_BasisFile}$ and basis information is provided (at least one of the optional arguments **Old Basis File**, **Insert File** and **Load File** is non-zero, see Section 11.2), \mathbf{f} and \mathbf{fstate} need not be set.

If $\mathbf{start} = \mathbf{Nag_Warm}$, \mathbf{f} and \mathbf{fstate} must be set (probably from a previous call). In this case $\mathbf{fstate}[i - 1]$ must be 0, 1, 2 or 3, for $i = 1, \dots, \mathbf{nf}$.

On exit: the final state of the variables. The elements of **fstate** have the following meaning:

fstate [$i - 1$]	State of the corresponding slack variable	Usual value of f [$i - 1$]
0	nonbasic	flow [$i - 1$]
1	nonbasic	fupp [$i - 1$]
2	superbasic	Between flow [$i - 1$] and fupp [$i - 1$]
3	basic	Between flow [$i - 1$] and fupp [$i - 1$]

Basic and superbasic slack variables may lead to the corresponding functions being outside their bounds by as much as the **Minor Feasibility Tolerance**.

Very occasionally some functions may be outside their bounds by as much as the **Minor Feasibility Tolerance**, and there may be some nonbasics for which **f**[$i - 1$] lies strictly between its bounds.

If **ninf** > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by **sinf** if scaling was not used).

Constraint: $0 \leq \mathbf{fstate}[i - 1] \leq 5$, for $i = 1, 2, \dots, \mathbf{nf}$.

30: **fmul**[**nf**] – double *Input/Output*

On entry: an estimate of γ , the vector of Lagrange-multipliers (shadow prices) for the constraints $l_F \leq F(x) \leq u_F$. All **nf** components must be defined. If nothing is known about γ , set **fmul**[$i - 1$] = 0.0, for $i = 1, 2, \dots, \mathbf{nf}$. For warm start use the values from a previous call.

On exit: the vector of the dual variables (Lagrange-multipliers) for the general constraints $l_F \leq F(x) \leq u_F$

31: **ns** – Integer * *Input/Output*

On entry: the number of superbasic variables. **ns** need not be specified for cold starts, but should retain its value from a previous call when warm start is used.

On exit: the final number of superbasic variables.

32: **ninf** – Integer * *Output*

33: **sinf** – double * *Output*

On exit: are the number and the sum of the infeasibilities of constraints that lie outside one of their bounds by more than the **Minor Feasibility Tolerance** before the solution is unscaled.

If the *linear* constraints are infeasible **x** minimizes the sum of the infeasibilities of the linear constraints subject to the upper and lower bounds being satisfied. In this case **ninf** gives the number of components of $A_L x$ lying outside their upper or lower bounds. The nonlinear constraints are not evaluated.

Otherwise, **x** minimizes the sum of infeasibilities of the *nonlinear* constraints subject to the linear constraints and upper and lower bounds being satisfied. In this case **ninf** gives the number of components of $F(x)$ lying outside their upper or lower bounds by more than the **Minor Feasibility Tolerance**. Again this is before the solution is unscaled.

34: **state** – Nag_E04State * *Communication Structure*

Note: **state** is a NAG defined type (see Section 2.2.1.1 of the Essential Introduction).

state contains internal information required for functions in this suite. It must not be modified in any way.

35: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 2.2.1.1 of the Essential Introduction).

36: **fail** – NagError *

Input/Output

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Internal error: memory allocation failed when attempting to allocate workspace sizes $\langle value \rangle$, $\langle value \rangle$ and $\langle value \rangle$.

NE_ALLOC_INSUFFICIENT

Internal memory allocation was insufficient. Please contact NAG.

NE_ARRAY_INPUT

Array element **igfun**[$\langle value \rangle$] = $\langle value \rangle$ is out of range 1 to **nf** = $\langle value \rangle$, or array element **jgvar**[$\langle value \rangle$] = $\langle value \rangle$ is out of range 1 to **n** = $\langle value \rangle$.

NE_BAD_PARAM

Basis file dimensions do not match this problem.

NE_BASIS_FAILURE

An error has occurred in the basis package. Check that arrays **iafun**, **javar**, **igfun** and **jgvar** contain values in the appropriate ranges and do not define duplicate elements of **a** or **g**. Set the optional argument **Print File** and examine the output carefully for further information.

NE_E04VGC_NOT_INIT

Initialization function `nag_opt_sparse_nlp_init` (e04vgc) has not been called.

NE_INT

On entry, **lena** < 1: **lena** = $\langle value \rangle$.

On entry, **leng** < 1: **leng** = $\langle value \rangle$.

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

On entry, **nea** < 0: **nea** = $\langle value \rangle$.

On entry, **neg** < 0: **neg** = $\langle value \rangle$.

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

NE_INT_2

On entry, **nfname** = $\langle value \rangle$, **nf** = $\langle value \rangle$.
Constraint: **nfname** = 1 or **nf**.

On entry, **nfname** is not equal to 1 or **n**. **nfname** = $\langle value \rangle$, **n** = $\langle value \rangle$.

On entry, **nxname** = $\langle value \rangle$, **n** = $\langle value \rangle$.
Constraint: **nxname** = 1 or **n**.

On entry, **nxname** is not equal to 1 or **n**. **nxname** = $\langle value \rangle$, **n** = $\langle value \rangle$.

On entry, **objrow** < 0 or **objrow** > **nf**. **objrow** = $\langle value \rangle$, **nf** = $\langle value \rangle$.

On entry, one but not both of **nxname** and **nfname** is equal to 1. **nxname** = $\langle value \rangle$, **nfname** = $\langle value \rangle$.

NE_INT_3

On entry, **nea** > **n** × **nf**. **nea** = ⟨value⟩, **n** = ⟨value⟩, **nf** = ⟨value⟩.

On entry, **neg** > **n** × **nf**. **neg** = ⟨value⟩, **n** = ⟨value⟩, **nf** = ⟨value⟩.

NE_INTERNAL_ERROR

An unexpected error has occurred. Set the optional argument **Print File** and examine the output carefully for further information.

NE_NOT_REQUIRED_ACC

The requested accuracy could not be achieved.

NE_NUM_DIFFICULTIES

User-supplied function computes incorrect constraint derivatives.

User-supplied function computes incorrect objective derivatives.

NE_REAL_2

On entry, bounds **flow** and **fupp** for ⟨value⟩ are equal and infinite. **flow** = **fupp** = ⟨value⟩, **bigbnd** = ⟨value⟩.

On entry, bounds **flow** and **fupp** for ⟨value⟩⟨value⟩ are equal and infinite. **flow** = **fupp** = ⟨value⟩, **bigbnd** = ⟨value⟩.

On entry, bounds for ⟨value⟩ are inconsistent. **flow** = ⟨value⟩, **fupp** = ⟨value⟩.

On entry, bounds for ⟨value⟩ are inconsistent. **xlow** = ⟨value⟩, **xupp** = ⟨value⟩.

On entry, bounds for ⟨value⟩⟨value⟩ are inconsistent. **flow** = ⟨value⟩, **fupp** = ⟨value⟩.

On entry, bounds for variable ⟨value⟩ are inconsistent. **xlow** = ⟨value⟩, **xupp** = ⟨value⟩.

On entry, bounds **xlow** and **xupp** for ⟨value⟩ are equal and infinite. **xlow** = **xupp** = ⟨value⟩, **bigbnd** = ⟨value⟩.

On entry, bounds **xlow** and **xupp** for variable ⟨value⟩ are equal and infinite. **xlow** = **xupp** = ⟨value⟩, **bigbnd** = ⟨value⟩.

NE_UNBOUNDED

The problem appears to be unbounded. The constraint violation limit has been reached.

The problem appears to be unbounded. The objective function is unbounded.

NE_USER_STOP

User-supplied function requested termination.

NE_USRFUN_UNDEFINED

Unable to proceed into undefined region of user-supplied function.

User-supplied function is undefined at the first feasible point.

User-supplied function is undefined at the initial point.

NW_NOT_FEASIBLE

The linear constraints appear to be infeasible.

The problem appears to be infeasible. Infeasibilities have been minimized.

The problem appears to be infeasible. Nonlinear infeasibilities have been minimized.

The problem appears to be infeasible. The linear equality constraints could not be satisfied.

NW_TOO_MANY_ITER

Iteration limit reached.

Major iteration limit reached.

Numerical difficulties have been encountered and no further progress can be made.

The superbasics limit is too small.

7 Accuracy

If the value of the optional argument **Major Optimality Tolerance** is set to 10^{-d} (default value = $\sqrt{\epsilon}$) and **fail.code** = **NE_NOERROR** on exit, then the final value of $f(x)$ should have approximately d correct significant digits.

8 Further Comments

This section describes the final output produced by `nag_opt_sparse_nlp_solve` (e04vhc). Intermediate and other output are given in Section 12.

8.1 The Final Output

Unless **Print File** = 0, the final output, including a listing of status of every variable and constraint will be sent to the **Print File** (see Section 11.2). The following describes the output for each constraint (row) and variable (column). A full stop (.) is printed for any numerical value that is zero.

The ROWS section

The i th constraint takes the form

$$\alpha \leq F_i x \leq \beta.$$

Internally, the constraints take the form $F(x) - s = 0$, where s is the set of slack variables (which happen to satisfy the bounds $\alpha \leq s \leq \beta$). For the i th constraint it is the slack variable s_i that is directly available, and it is sometimes convenient to refer to its state. A ‘.’ is printed for any numerical value that is exactly zero.

Label	Description
Number	is the value of $n + i$. (This is used internally to refer to s_i in the intermediate output.)
Row	gives the name of F_i .
State	the state of s_i (the state of F_i relative to the bounds α and β). The various states possible are as follows: <ul style="list-style-type: none"> LL s_i is nonbasic at its lower limit, α. UL s_i is nonbasic at its upper limit, β. EQ s_i is nonbasic and fixed at the value $\alpha = \beta$. FR s_i is nonbasic and currently zero, even though it is free to take any value between its bounds α and β. BS s_i is basic. SBS s_i is superbasic. <p>A key is sometimes printed before State to give some additional information about the state of a variable. Note that unless the optional argument Scale Option = 0 (see Section 11.2) is specified, the tests for assigning a key are applied to the variables of the scaled problem.</p> <ul style="list-style-type: none"> A <i>Alternative optimum possible</i>. The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to

start moving away from its bound, there would be no change in the value of 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 basic or superbasic, but it is equal to (or very close to) one of its bounds.
- I *Infeasible.* The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the optional argument **Feasibility Tolerance** (see Section 11.2).
- N *Not precisely optimal.* The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional argument **Major Optimality Tolerance** (see Section 11.2), the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

Activity	is the value of F_i at the final iterate.
Slack Activity	is the value by which the row differs from its nearest bound. (For the free row (if any), it is set to Activity.)
Lower Limit	is α , the lower bound specified for the variable s_i . None indicates that $\text{flow}[j - 1] \leq -\text{bigbnd}$.
Upper Bound	is β , the upper bound specified for the variable s_i . None indicates that $\text{fupp}[j - 1] \geq \text{bigbnd}$.
Dual Activity	is the value of the dual variable π_i (the Lagrange-multiplier for the i th constraint). The full vector π always satisfies $B^T \pi = g_B$, where B is the current basis matrix and g_B contains the associated gradients for the current objective function. For FP problems, π_i is set to zero.
i	gives the index i of the i th row.

The COLUMNS section

Let the j th component of x be the variable x_j and assume that it satisfies the bounds $\alpha \leq x_j \leq \beta$. A ‘.’ is printed for any numerical value that is exactly zero.

Label	Description
Number	is the column number j . (This is used internally to refer to x_j in the intermediate output.)
Column	gives the name of x_j .
State	the state of x_j relative to the bounds α and β . The various states possible are as follows:
LL	x_j is nonbasic at its lower limit, α .
UL	x_j is nonbasic at its upper limit, β .
EQ	x_j is nonbasic and fixed at the value $\alpha = \beta$.
FR	x_j is nonbasic and currently zero, even though it is free to take any value between its bounds α and β .
BS	x_j is basic.
SBS	x_j is superbasic.

A key is sometimes printed before State to give some additional information about the state of a variable. Note that unless the optional argument **Scale Option** = 0

(see Section 11.2) is specified, the tests for assigning a key are applied to the variables of the scaled problem.

- A *Alternative optimum possible.* The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of 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 basic or superbasic, but it is equal to (or very close to) one of its bounds.
- I *Infeasible.* The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the optional argument **Feasibility Tolerance** (see Section 11.2).
- N *Not precisely optimal.* The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional argument **Major Optimality Tolerance** (see Section 11.2), the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

Activity	is the value of x_j at the final iterate.
Obj Gradient	is the value of g_j at the final iterate. For FP problems, g_j is set to zero.
Lower Bound	is the lower bound specified for the variable. None indicates that $\mathbf{xlow}[j - 1] \leq -bigbnd$.
Upper Bound	is the upper bound specified for the variable. None indicates that $\mathbf{xupp}[j - 1] \geq bigbnd$.
Reduced Gradnt	is the value of the reduced gradient $d_j = g_j - \pi^T a_j$ where a_j is the j th column of the constraint matrix. For FP problems, d_j is set to zero.
$m + j$	is the value of $m + j$.

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.

9 Example

This is a reformulation of Problem 74 from Hock and Schittkowsky (1981) and involves the minimization of the nonlinear function

$$f(x) = 10^{-6}x_3^3 + \frac{2}{3} \times 10^{-6}x_4^3 + 3x_3 + 2x_4$$

subject to the bounds

$$\begin{aligned} -0.55 &\leq x_1 \leq 0.55, \\ -0.55 &\leq x_2 \leq 0.55, \\ 0 &\leq x_3 \leq 1200, \\ 0 &\leq x_4 \leq 1200, \end{aligned}$$

to the nonlinear constraints

$$\begin{aligned} 1000 \sin(-x_1 - 0.25) &+ 1000 \sin(-x_2 - 0.25) - x_3 &= &- 894.8, \\ 1000 \sin(x_1 - 0.25) &+ 1000 \sin(x_1 - x_2 - 0.25) - x_4 &= &- 894.8, \\ 1000 \sin(x_2 - 0.25) &+ 1000 \sin(x_2 - x_1 - 0.25) &= &- 1294.8, \end{aligned}$$

and to the linear constraints

$$\begin{aligned} -x_1 + x_2 &\geq -0.55, \\ x_1 - x_2 &\geq -0.55. \end{aligned}$$

The initial point, which is infeasible, is

$$x_0 = (0, 0, 0, 0)^T,$$

and $f(x_0) = 0$.

The optimal solution (to five figures) is

$$x^* = (0.11887, -0.39623, 679.94, 1026.0)^T,$$

and $f(x^*) = 5126.4$. All the nonlinear constraints are active at the solution.

The example in the document for `nag_opt_sparse_nlp_jacobian` (e04vjc) solves the above problem. It first calls `nag_opt_sparse_nlp_jacobian` (e04vjc) to determine the sparsity pattern before calling `nag_opt_sparse_nlp_solve` (e04vhc).

The example in the document for `nag_opt_sparse_nlp_option_set_file` (e04vkc) solves the above problem using some of the optional arguments described in Section 11.

The formulation of the problem combines the constraints and the objective into a single vector (F) which is split into linear part ($A_L x$) and a nonlinear part (f). For example we could write

$$F = \begin{pmatrix} 1000 \sin(-x_1 - 0.25) + 1000 \sin(-x_2 - 0.25) - x_3 \\ 1000 \sin(x_1 - 0.25) + 1000 \sin(x_1 - x_2 - 0.25) - x_4 \\ 1000 \sin(x_2 - 0.25) + 1000 \sin(x_2 - x_1 - 0.25) \\ -x_1 + x_2 \\ x_1 - x_2 \\ 10^{-6}x_3^3 + \frac{2}{3} \times 10^{-6}x_4^3 + 3x_3 + 2x_4 \end{pmatrix} = f + A_L x$$

where

$$f = \begin{pmatrix} 1000 \sin(-x_1 - 0.25) + 1000 \sin(-x_2 - 0.25) \\ 1000 \sin(x_1 - 0.25) + 1000 \sin(x_1 - x_2 - 0.25) \\ 1000 \sin(x_2 - 0.25) + 1000 \sin(x_2 - x_1 - 0.25) \\ 0 \\ 0 \\ 10^{-6}x_3^3 + \frac{2}{3} \times 10^{-6}x_4^3 \end{pmatrix}$$

and

$$A_L = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 3 & 2 \end{pmatrix}.$$

The non-zero elements of A_L need to be stored in the triples (**iafun**[$k-1$], **javar**[$k-1$], **a**[$k-1$]) in any order. For example

k	1	2	3	4	5	6	7	8
iafun [$k-1$]	1	2	4	4	5	5	6	6
javar [$k-1$]	3	4	1	2	1	2	3	4
a [$k-1$]	-1	-1	-1	1	1	-1	3	2

The nonlinear functions f and the Jacobian need to be supplied in **usrfun**. Please note that there is no need to assign any value to f_4 or f_5 as there is no nonlinear part in F_4 or F_5 .

The non-zero entries of the Jacobian of f are

$$\begin{aligned}\frac{\partial f_1}{\partial x_1} &= -1000 \cos(-x_1 - 0.25) \\ \frac{\partial f_1}{\partial x_2} &= -1000 \cos(-x_2 - 0.25) \\ \frac{\partial f_2}{\partial x_1} &= 1000 \cos(x_1 - 0.25) + 1000 \cos(x_1 - x_2 - 0.25) \\ \frac{\partial f_2}{\partial x_2} &= -1000 \cos(x_1 - x_2 - 0.25) \\ \frac{\partial f_3}{\partial x_1} &= -1000 \cos(x_2 - x_1 - 0.25) \\ \frac{\partial f_3}{\partial x_2} &= 1000 \cos(x_2 - 0.25) + 1000 \cos(x_2 - x_1 - 0.25) \\ \frac{\partial f_6}{\partial x_3} &= 3 \times 10^{-6} x_3^2 \\ \frac{\partial f_6}{\partial x_4} &= 2 \times 10^{-6} x_4^2\end{aligned}$$

So the arrays **igfun** and **jgvar** must contain:

k	1	2	3	4	5	6	7	8
igfun [$k - 1$]	1	1	2	2	3	3	6	6
jgvar [$k - 1$]	1	2	1	2	1	2	3	4

and **usrfun** must return in $\mathbf{g}[k - 1] = \frac{\partial f_i}{\partial x_j}$ where $i = \mathbf{igfun}[k - 1]$ and $j = \mathbf{jgvar}[k - 1]$.

9.1 Program Text

```
/* nag_nag_opt_sparse_nlp_solve (e04vhc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 8, 2004.
 */

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

static void usrfun(Integer *status, Integer n, const double x[],
                  Integer needf, Integer nf, double f[], Integer needg,
                  Integer leng, double g[], Nag_Comm *comm);

int main(void)
{
    /* Scalars */
    double objadd, sinf;
    Integer exit_status, i, lena, leng, n, nea, neg, nf, nfname, ninf, ns, nxname;
    Integer objrow, start_int;

    /* Arrays */
    char **fnames=0, **xnames=0;
    char prob[9];
    double *a=0, *f=0, *flow=0, *fmul=0, *fupp=0, *x=0, *xlow=0, *xmul=0, *xupp=0;
```

```

Integer *fstate=0, *iafun=0, *igfun=0, *iw=0, *javar=0, *jgvar=0, *xstate=0;

/*Nag Types*/
Nag_E04State state;
NagError fail;
Nag_Start start;
Nag_Comm comm;
Nag_FileID fileid;

exit_status = 0;
INIT_FAIL(fail);
Vprintf("nag_opt_sparse_nlp_solve (e04vhc) Example Program Results\n");

/* Skip heading in data file */
Vscanf("%*[\n] ");
Vscanf("%ld%ld%*[\n] ", &n, &nf);

Vscanf("%ld%ld%ld%ld%*[\n] ",
        &nea, &neg, &objrow, &start_int);

if (n > 0 && nf > 0 && nea >= 0 && neg >= 0)
{
    if (start_int == 0)
    {
        start = Nag_Cold;
    }
    else
    {
        start = Nag_Warm;
    }
    nxname = n;
    nfname = nf;
    lena = MAX(1,nea);
    leng = MAX(1,neg);
    /* Allocate memory */
    if ( !(fnames = NAG_ALLOC(nfname, char *)) ||
        !(xnames = NAG_ALLOC(nxname, char *)) ||
        !(a = NAG_ALLOC(lena, double)) ||
        !(f = NAG_ALLOC(nf, double)) ||
        !(flow = NAG_ALLOC(nf, double)) ||
        !(fmul = NAG_ALLOC(nf, double)) ||
        !(fupp = NAG_ALLOC(nf, double)) ||
        !(x = NAG_ALLOC(n, double)) ||
        !(xlow = NAG_ALLOC(n, double)) ||
        !(xmul = NAG_ALLOC(n, double)) ||
        !(xupp = NAG_ALLOC(n, double)) ||
        !(fstate = NAG_ALLOC(nf, Integer)) ||
        !(iafun = NAG_ALLOC(lena, Integer)) ||
        !(igfun = NAG_ALLOC(leng, Integer)) ||
        !(javar = NAG_ALLOC(lena, Integer)) ||
        !(jgvar = NAG_ALLOC(leng, Integer)) ||
        !(xstate = NAG_ALLOC(n, Integer)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    Vprintf("Invalid n or nf or nea or neg\n");
    exit_status = 1;
    goto END;
}
objadd = 0.;
strcpy(prob, "");

/* Read the variable names xnames */
for (i = 1; i <= nxname; ++i)
{
    xnames[i-1] = NAG_ALLOC(9, char);
}

```

```

    Vscanf(" ' %8s '", xnames[i-1]);
  }
Vscanf("%*[\n] ");

/* Read the function names fnames */
for (i = 1; i <= nfname; ++i)
  {
    fnames[i - 1] = NAG_ALLOC(9, char);
    Vscanf(" ' %8s'", fnames[i-1]);
  }
Vscanf("%*[\n] ");

/* Read the sparse matrix a, the linear part of f */
for (i = 1; i <= nea; ++i)
  {
    /* For each element read row, column, A(row,column) */
    Vscanf("%ld%ld%lf%*[\n] ", &iafun[i - 1], &javar[i - 1],
          &a[i - 1]);
  }
/* Read the structure of sparse matrix G, the nonlinear part of f */
for (i = 1; i <= neg; ++i)
  {
    /* For each element read row, column */
    Vscanf("%ld%ld%*[\n] ", &igfun[i - 1], &jgvar[i - 1]);
  }

/* Read the lower and upper bounds on the variables */
for (i = 1; i <= n; ++i)
  {
    Vscanf("%lf%lf%*[\n] ", &xlow[i - 1], &xupp[i - 1]);
  }

/* Read the lower and upper bounds on the functions */
for (i = 1; i <= nf; ++i)
  {
    Vscanf("%lf%lf%*[\n] ", &flow[i - 1], &fupp[i - 1]);
  }

/* Initialise x, xstate, xmul, f, fstate, fmul */
for (i = 1; i <= n; ++i)
  {
    Vscanf("%lf", &x[i - 1]);
  }
Vscanf("%*[\n] ");

for (i = 1; i <= n; ++i)
  {
    Vscanf("%ld", &xstate[i - 1]);
  }
Vscanf("%*[\n] ");

for (i = 1; i <= n; ++i)
  {
    Vscanf("%lf", &xmul[i - 1]);
  }
Vscanf("%*[\n] ");

for (i = 1; i <= nf; ++i)
  {
    Vscanf("%lf", &f[i - 1]);
  }
Vscanf("%*[\n] ");

for (i = 1; i <= nf; ++i)
  {
    Vscanf("%ld", &fstate[i - 1]);
  }

```

```

Vscanf("%*[\n] ");

for (i = 1; i <= nf; ++i)
{
    Vscanf("%lf", &fmul[i - 1]);
}
Vscanf("%*[\n] ");

/* Call nag_opt_sparse_nlp_init (e04vgc) to initialise e04vhf. */
/* nag_opt_sparse_nlp_init (e04vgc).
 * Initialization function for nag_opt_sparse_nlp_solve
 * (e04vhc)
 */
nag_opt_sparse_nlp_init(&state, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Initialisation of nag_opt_sparse_nlp_init (e04vgc) failed.\n");
    exit_status = 1;
    goto END;
}

/* By default e04vhf does not print monitoring */
/* information. Call nag_open_file (x04acc) to set the print file fileid */
/* nag_open_file (x04acc).
 * Open unit number for reading, writing or appending, and
 * associate unit with named file
 */
nag_open_file("", 2, &fileid, &fail);

/* nag_opt_sparse_nlp_option_set_integer (e04vmc).
 * Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
 * from an integer argument
 */
nag_opt_sparse_nlp_option_set_integer("Print file", fileid, &state, &fail);

/* Solve the problem. */
/* nag_opt_sparse_nlp_solve (e04vhc).
 * General sparse nonlinear optimizer
 */
nag_opt_sparse_nlp_solve(start, nf, n, nxname, nfname, objadd, objrow, prob,
                        usrfun, iafun, javar, a, lena, nea, igfun, jgvar,
                        leng, neg, xlow, xupp, xnames, flow, fupp, fnames, x,
                        xstate, xmul, f, fstate, fmul, &ns, &ninf, &sinf,
                        &state, &comm, &fail);

if (fail.code == NE_NOERROR)
{
    Vprintf("Final objective value = %11.1f\n", f[objrow - 1]);
    Vprintf("Optimal X = ");

    for (i = 1; i <= n; ++i)
    {
        Vprintf("%9.2f%s", x[i - 1], i%7 == 0 || i == n ? "\n": " ");
    }
}
else
{
    Vprintf ("Error message from nag_opt_sparse_nlp_solve (e04vhc) %s\n",
            fail.message);
}

END:
for (i=0; i < nxname; i++)
{
    NAG_FREE(xnames[i]);
}
for (i=0; i < nfname; i++)
{
    NAG_FREE(fnames[i]);
}

```

```

if (fnames) NAG_FREE(fnames);
if (xnames) NAG_FREE(xnames);
if (a) NAG_FREE(a);
if (f) NAG_FREE(f);
if (flow) NAG_FREE(flow);
if (fmul) NAG_FREE(fmul);
if (fupp) NAG_FREE(fupp);
if (x) NAG_FREE(x);
if (xlow) NAG_FREE(xlow);
if (xmul) NAG_FREE(xmul);
if (xupp) NAG_FREE(xupp);
if (fstate) NAG_FREE(fstate);
if (iafun) NAG_FREE(iafun);
if (igfun) NAG_FREE(igfun);
if (iw) NAG_FREE(iw);
if (javar) NAG_FREE(javar);
if (jgvar) NAG_FREE(jgvar);
if (xstate) NAG_FREE(xstate);

return exit_status;
}

static void usrfun(Integer *status, Integer n, const double x[],
                  Integer needf, Integer nf, double f[], Integer needg,
                  Integer leng, double g[], Nag_Comm *comm)
{
    /* Parameter adjustments */
#define X(I) x[(I)-1]
#define F(I) f[(I)-1]
#define G(I) g[(I)-1]

    /* Function Body */
    if (needf > 0)
    {
        /* The nonlinear components of F_i(x) need to be assigned, */
        /* for i = 1 to nf */
        F(1) = sin(-X(1) - .25) * 1e3 + sin(-X(2) - .25) * 1e3;
        F(2) = sin(X(1) - .25) * 1e3 + sin(X(1) - X(2) - .25) * 1e3;
        F(3) = sin(X(2) - X(1) - .25) * 1e3 + sin(X(2) - .25) * 1e3;
        /* N.B. in this example there is no need to assign for the wholly */
        /* linear components f_4(x) and f_5(x). */
        F(6) = X(3) * (X(3) * X(3)) * 1e-6 + X(4) * (X(4) * X(4)) * 2e-6 / 3.;
    }

    if (needg > 0)
    {
        /* The derivatives of the function F_i(x) need to be assigned.
        * G(k) should be set to partial derivative df_i(x)/dx_j where
        * i = IGFUN(k) and j = IGVAR(k), for k = 1 to leng.
        */
        G(1) = cos(-X(1) - .25) * -1e3;
        G(2) = cos(-X(2) - .25) * -1e3;
        G(3) = cos(X(1) - .25) * 1e3 + cos(X(1) - X(2) - .25) * 1e3;
        G(4) = cos(X(1) - X(2) - .25) * -1e3;
        G(5) = cos(X(2) - X(1) - .25) * -1e3;
        G(6) = cos(X(2) - X(1) - .25) * 1e3 + cos(X(2) - .25) * 1e3;
        G(7) = X(3) * X(3) * 3e-6;
        G(8) = X(4) * X(4) * 2e-6;
    }

    return;
} /* usrfun */

```

9.2 Program Data

nag_opt_sparse_nlp_solve (e04vhc) Example Program Data

```

4 6 : Values of N and NF
8 8 6 0 : Values of NEA, NEG, OBJROW and START

'X1      ' 'X2      ' 'X3      ' 'X4      ' : XNAMES
'NlnCon_1' 'NlnCon_2' 'NlnCon_3' 'LinCon_1' 'LinCon_2' 'Objectiv' : FNAMES

1 3 -1.0E0 : Nonzero elements of sparse matrix A, the linear part of F.
2 4 -1.0E0 : Each row IAFUN(i), JAVAR(i), A(IAFUN(i),JAVAR(i)), i = 1 to NEA
4 1 -1.0E0
4 2 1.0E0
5 1 1.0E0
5 2 -1.0E0
6 3 3.0E0
6 4 2.0E0

1 1 : Nonzero row/column structure of G, IGFUN(i), JGVAR(i), i = 1 to NEG
1 2
2 1
2 2
3 1
3 2
6 3
6 4

-0.55E0 0.55E0 : Bounds on the variables, XLOW(i), XUPP(i), for i = 1 to N
-0.55E0 0.55E0
0.0E0 1200.0E0
0.0E0 1200.0E0

-894.8E0 -894.8E0 : Bounds on the functions, FLOW(i), FUPP(i), for i = 1 to NF
-894.8E0 -894.8E0
-1294.8E0 -1294.8E0
-0.55E0 1.0E25
-0.55E0 1.0E25
-1.0E25 1.0E25

0.0 0.0 0.0 0.0 : Initial values of X(i), for i = 1 to N
0 0 0 0 : Initial values of XSTATE(i), for i = 1 to N
0.0 0.0 0.0 0.0 : Initial values of XMUL(i), for i = 1 to N

0.0 0.0 0.0 0.0 0.0 0.0 : Initial values of F(i), for i = 1 to NF
0 0 0 0 0 0 : Initial values of FSTATE(i), for i = 1 to NF
0.0 0.0 0.0 0.0 0.0 0.0 : Initial values of FMUL(i), for i = 1 to NF

```

9.3 Program Results

nag_opt_sparse_nlp_solve (e04vhc) Example Program Results

```

Parameters
=====

Files
-----
Solution file..... 0      Old basis file ..... 0      (Print file)..... 6
Insert file..... 0      New basis file ..... 0      (Summary file)..... 0
Punch file..... 0      Backup basis file..... 0
Load file..... 0      Dump file..... 0

Frequencies
-----
Print frequency..... 100    Check frequency..... 60    Save new basis map..... 100
Summary frequency..... 100    Factorization frequency 50    Expand frequency..... 10000

QP subproblems
-----
QP solver Cholesky.....
Scale tolerance..... 0.900    Minor feasibility tol.. 1.00E-06    Iteration limit..... 10000

```

Scale option.....	0	Minor optimality tol..	1.00E-06	Minor print level.....	1
Crash tolerance.....	0.100	Pivot tolerance.....	2.05E-11	Partial price.....	1
Crash option.....	3	Elastic weight.....	1.00E+04	Prtl price section (A)	4
		New superbasics.....	99	Prtl price section (-I)	6

The SQP Method

Minimize.....		Cold start.....		Proximal Point method..	1
Nonlinear objectiv vars	4	Objective Row.....	6	Function precision.....	1.72E-13
Unbounded step size....	1.00E+20	Superbasics limit.....	4	Difference interval....	4.15E-07
Unbounded objective....	1.00E+15	Reduced Hessian dim....	4	Central difference int.	5.57E-05
Major step limit.....	2.00E+00	Derivative linesearch..		Derivative option.....	1
Major iterations limit.	1000	Linesearch tolerance...	0.90000	Verify level.....	0
Minor iterations limit.	500	Penalty parameter.....	0.00E+00	Major Print Level.....	1
		Major optimality tol...	2.00E-06		

Hessian Approximation

Full-Memory Hessian....		Hessian updates.....	99999999	Hessian frequency.....	99999999
				Hessian flush.....	99999999

Nonlinear constraints

Nonlinear constraints..	3	Major feasibility tol..	1.00E-06	Violation limit.....	1.00E+06
Nonlinear Jacobian vars	2				

Miscellaneous

LU factor tolerance....	3.99	LU singularity tol.....	2.05E-11	Timing level.....	0
LU update tolerance....	3.99	LU swap tolerance.....	1.03E-04	Debug level.....	0
LU partial pivoting...		eps (machine precision)	1.11E-16	System information....	No

Nonlinear constraints	3	Linear constraints	3
Nonlinear variables	4	Linear variables	0
Jacobian variables	2	Objective variables	4
Total constraints	6	Total variables	4

The user has defined 8 out of 8 first derivatives

Cheap test of user-supplied problem derivatives...

The constraint gradients seem to be OK.

--> The largest discrepancy was 2.23E-08 in constraint 7

The objective gradients seem to be OK.

Gradient projected in one direction 0.0000000000E+00
 Difference approximation 4.49060460280E-21

Itns	Major	Minors	Step	nCon	Feasible	Optimal	MeritFunction	L+U	BSwap	nS	condHz	Penalty
3	0	3		1	8.0E+02	1.0E-00	0.0000000E+00	17		1	1.7E+07	_ r
4	1	1	1.2E-03	2	4.0E+02	9.9E-01	9.6317131E+05	16		1	4.8E+06	2.8E+00 _n rl
5	2	1	1.3E-03	3	2.7E+02	5.5E-01	9.6122945E+05	16				2.8E+00 _s 1
5	3	0	7.5E-03	4	8.8E+01	5.4E-01	9.4691061E+05	16				2.8E+00 _ 1
5	4	0	2.3E-02	5	2.9E+01	5.3E-01	9.0468403E+05	16				2.8E+00 _ 1
5	5	0	6.9E-02	6	8.9E+00	5.0E-01	7.8452897E+05	16				2.8E+00 _ 1
6	6	1	2.2E-01	7	2.3E+00	5.5E+01	4.8112339E+05	16		1	8.7E+03	2.8E+00 _ 1
7	7	1	8.3E-01	8	1.7E-01	4.2E+00	2.6898257E+04	16		1	7.6E+03	2.8E+00 _ 1
8	8	1	1.0E+00	9	1.8E-02	8.7E+01	6.2192920E+03	15	1	1	1.2E+02	2.8E+00 _
9	9	1	1.0E+00	10	1.7E-02	7.9E+00	5.4526185E+03	15		1	9.4E+01	2.8E+00 _
10	10	1	1.0E+00	11	1.7E-04	9.6E-01	5.1266089E+03	15		1	1.0E+02	2.8E+00 _
11	11	1	1.0E+00	12	1.7E-06	5.8E-02	5.1264988E+03	15		1	9.5E+01	2.8E+00 _
12	12	1	1.0E+00	13	(1.2E-08)	6.9E-05	5.1264981E+03	15		1	9.5E+01	2.8E+00 _
13	13	1	1.0E+00	14	(6.7E-15)	(3.0E-09)	5.1264981E+03	15		1	9.5E+01	6.0E+00 _


```
E04VHF EXIT  0 -- finished successfully
E04VHF INFO  1 -- optimality conditions satisfied
```

```
Problem name
No. of iterations      13  Objective value      5.1264981096E+03
No. of major iterations 13  Linear objective      4.0919702248E+03
Penalty parameter      6.029E+00  Nonlinear objective  1.0345278848E+03
No. of calls to funobj  15  No. of calls to funcon  15
No. of superbasics     1  No. of basic nonlinear  3
No. of degenerate steps 0  Percentage            0.00
Max x                   4 1.0E+03  Max pi                3 5.5E+00
Max Primal infeas      0 0.0E+00  Max Dual infeas      1 4.6E-08
Nonlinear constraint violn 5.7E-12
```

```
Name                                Objective Value      5.1264981096E+03
```

```
Status      Optimal Soln      Iteration    13      Superbasics    1
```

```
Objective      (Min)
```

```
RHS
```

```
Ranges
```

```
Bounds
```

```
Section 1 - Rows
```

Number	...Row..	State	...Activity...	Slack Activity	..Lower Limit.	..Upper Limit.	..Dual Activity	..i
5	NlnCon_1	EQ	-894.80000	0.00000	-894.80000	-894.80000	-4.38698	1
6	NlnCon_2	EQ	-894.80000	0.00000	-894.80000	-894.80000	-4.10563	2
7	NlnCon_3	EQ	-1294.80000	0.00000	-1294.80000	-1294.80000	-5.46328	3
8	LinCon_1	BS	-0.51511	0.03489	-0.55000	None	.	4
9	LinCon_2	BS	0.51511	1.06511	-0.55000	None	.	5
10	Objectiv	BS	4091.97022	4091.97022	None	None	-1.0	6

```
Section 2 - Columns
```

Number	.Column.	State	...Activity...	.Obj Gradient.	..Lower Limit.	..Upper Limit.	Reduced Gradnt	m+j
1	X1	BS	0.11888	.	-0.55000	0.55000	-0.00000	7
2	X2	BS	-0.39623	.	-0.55000	0.55000	0.00000	8
3	X3	SBS	679.94532	4.38698	.	1200.00000	0.00000	9
4	X4	BS	1026.06713	4.10563	.	1200.00000	-0.00000	10

Final objective value = 5126.5
Optimal X = 0.12 -0.40 679.95 1026.07

Note: the remainder of this document is intended for more advanced users. Section 10 contains a detailed algorithm description that may be needed in order to understand Sections 11 and 12. Section 11 describes the optional arguments that may be set by calls to `nag_opt_sparse_nlp_option_set_file` (e04vkc), `nag_opt_sparse_nlp_option_set_string` (e04vlc), `nag_opt_sparse_nlp_option_set_integer` (e04vmc) and/or `nag_opt_sparse_nlp_option_set_double` (e04vnc). Section 12 describes the quantities that can be requested to monitor the course of the computation.

10 Algorithmic Details

Here we summarize the main features of the SQP algorithm used in `nag_opt_sparse_nlp_solve` (e04vhc) and introduce some terminology used in the description of the function and its arguments. The SQP algorithm is fully described in Gill *et al.* (2002).

10.1 Constraints and Slack Variables

The upper and lower bounds on the components of $f(x)$ and $A_L x$ are said to define the *general constraints* of the problem. `nag_opt_sparse_nlp_solve` (e04vhc) converts the general constraints to equalities by introducing a set of *slack variables* $s = (s_1, s_2, \dots, s_m)^T$. For example, the linear constraint

$5 \leq 3x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. The minimization problem (1) can therefore be written in the equivalent form

$$\underset{x,s}{\text{minimize}} f_0(x) \quad \text{subject to} \quad \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \quad (3)$$

The general constraints become the equalities $f(x) - s_N = 0$ and $A_L x - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

10.2 Major Iterations

The basic structure of the SQP algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate a QP subproblem is used to generate a search direction towards the next iterate x_{k+1} . The constraints of the subproblem are formed from the linear constraints $A_L x - s_L = 0$ and the nonlinear constraint linearization

$$f(x_k) + f'(x_k)(x - x_k) - s_N = 0, \quad (4)$$

where $f'(x_k)$ denotes the *Jacobian matrix*, whose elements are the first derivatives of $f(x)$ evaluated at x_k . The QP constraints therefore comprise the m linear constraints

$$\begin{array}{rcl} f'(x_k)x & -s_N & = -f(x_k) + f'(x_k)x_k, \\ A_L x & -s_L & = 0, \end{array} \quad (5)$$

where x and s are bounded above and below by u and l as before. If the m by n matrix A and m -vector b are defined as

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix}, \quad (6)$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} q(x) \quad \text{subject to} \quad Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \quad (7)$$

where $q(x)$ is a quadratic approximation to a modified Lagrangian function (see Gill *et al.* (2002)).

10.3 Minor Iterations

Solving the QP subproblem is itself an iterative procedure. The iterations of the QP solver are the *minor* iterations of the SQP method. At each minor iteration, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = b, \quad (8)$$

where the *basic matrix* B is square and nonsingular. The elements of x_B , x_S and x_N are called the *basic*, *superbasic* and *nonbasic* variables respectively; they are a permutation of the elements of x and s . At a QP subproblem, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will normally be equal to one of their bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or the sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. The number of superbasic variables (n_S , say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero for LP problems.

If it appears that no improvement can be made with the current definition of B , S and N , a nonbasic variable is selected to be added to S , and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints $Ax - s = b$ are the *dual variables* π . Similarly, each variable in (x, s) has an associated *reduced gradient* d_j . The reduced gradients for the variables x are the quantities $g - A^T \pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks

are the dual variables π . The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics. In practice, an *approximate* QP solution $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ is found by relaxing these conditions.

10.4 The Merit Function

After a QP subproblem has been solved, new estimates of the solution are computed using a line search on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f_0(x) - \pi^T(f(x) - s_N) + \frac{1}{2}(f(x) - s_N)^T D(f(x) - s_N), \quad (9)$$

where D is a diagonal matrix of penalty arguments ($D_{ii} \geq 0$). If (x_k, s_k, π_k) denotes the current solution estimate and $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ denotes the QP solution, the line search determines a step α_k ($0 < \alpha_k \leq 1$) such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \hat{x}_k - x_k \\ \hat{s}_k - s_k \\ \hat{\pi}_k - \pi_k \end{pmatrix} \quad (10)$$

gives a *sufficient decrease* in the merit function (see (9)). When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for \mathcal{M} (see Gill *et al.* (1992)). s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem (see Gill *et al.* (1986c) and Eldersveld (1991)).

10.5 Treatment of Constraint Infeasibilities

`nag_opt_sparse_nlp_solve` (e04vhc) makes explicit allowance for infeasible constraints. First, infeasible *linear* constraints are detected by solving the linear program

$$\underset{x, v, w}{\text{minimize}} e^T(v + w) \quad \text{subject to } l \leq \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \leq u, \quad u \geq 0, \quad w \geq 0, \quad (11)$$

where e is a vector of ones, and the nonlinear constraint bounds are temporarily excluded from l and u . This is equivalent to minimizing the sum of the general linear constraint violations subject to the bounds on x . (The sum is the ℓ_1 -norm of the linear constraint violations. In the linear programming literature, the approach is called *elastic programming*.)

The linear constraints are infeasible if the optimal solution of (11) has $v \neq 0$ or $w \neq 0$. `nag_opt_sparse_nlp_solve` (e04vhc) then terminates without computing the nonlinear functions.

Otherwise, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) `nag_opt_sparse_nlp_solve` (e04vhc) proceeds to solve nonlinear problems as given, using search directions obtained from the sequence of QP subproblems (see (7)).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variable π for the nonlinear constraints become large), `nag_opt_sparse_nlp_solve` (e04vhc) enters ‘elastic’ mode and thereafter solves the problem

$$\underset{x, v, w}{\text{minimize}} f_0(x) + \gamma e^T(v + w) \quad \text{subject to } l \leq \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \quad (12)$$

where γ is a non-negative argument (the *elastic weight*), and $f_0(x) + \gamma e^T(v + w)$ is called a *composite objective* (the ℓ_1 penalty function for the nonlinear constraints).

The value of γ may increase automatically by multiples of 10 if the optimal u and w continue to be non-zero. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds.

The initial value of γ is controlled by the optional arguments **Elastic Mode** and **Elastic Weight** (see Section 11.2).

11 Optional Arguments

Several optional arguments in `nag_opt_sparse_nlp_solve` (e04vhc) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of `nag_opt_sparse_nlp_solve` (e04vhc) 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. A complete list of optional arguments and their default values is given in Section 11.1.

Optional arguments may be specified by calling one, or more, of the functions `nag_opt_sparse_nlp_option_set_file` (e04vkc), `nag_opt_sparse_nlp_option_set_string` (e04vlc), `nag_opt_sparse_nlp_option_set_integer` (e04vmc) and `nag_opt_sparse_nlp_option_set_double` (e04vnc) prior to a call to `nag_opt_sparse_nlp_solve` (e04vhc).

`nag_opt_sparse_nlp_option_set_file` (e04vkc) 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
  Print Level = 5
End
```

The call

```
e04vkc (ioptns, &state, &fail);
```

can then be used to read the file on file descriptor `ioptns` which can be obtained by a prior call to `nag_open_file` (x04acc). **fail.code = NE_NOERROR** on successful exit. `nag_opt_sparse_nlp_option_set_file` (e04vkc) should be consulted for a full description of this method of supplying optional arguments.

`nag_opt_sparse_nlp_option_set_string` (e04vlc), `nag_opt_sparse_nlp_option_set_integer` (e04vmc) and `nag_opt_sparse_nlp_option_set_double` (e04vnc) can be called to supply options directly, one call being necessary for each optional argument. For example,

```
e04vlc ("Print Level = 5", &state, &fail);
```

`nag_opt_sparse_nlp_option_set_string` (e04vlc), `nag_opt_sparse_nlp_option_set_integer` (e04vmc) and `nag_opt_sparse_nlp_option_set_double` (e04vnc) should be consulted for a full description of this method of supplying optional arguments.

All optional arguments not specified by you are set to their default values. Optional arguments specified by you are unaltered by `nag_opt_sparse_nlp_solve` (e04vhc) (unless they define invalid values) and so remain in effect for subsequent calls to `nag_opt_sparse_nlp_solve` (e04vhc), unless altered by you.

11.1 Optional Argument Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 11.2. The minimum abbreviation of each keyword is underlined. If no characters of an optional qualifier are underlined, the qualifier may be omitted. The letter *a* denotes a phrase (character string) that qualifies an option. The letters *i* and *r* denote Integer and double values required with certain options. The number ϵ is a generic notation for *machine precision* (see `nag_machine_precision` (X02AJC)), and ϵ_R denotes the relative precision of the objective function (see **Function Precision**).

Optional arguments used to specify files (e.g., **Dump File** and **Print File**) have type **Nag_FileID**. This ID value must either be set to 0 (the default value) in which case there will be no output, or will be as returned by a call of `nag_open_file` (x04acc).

Optional Arguments	Default Values
<u>B</u> ackup <u>B</u> asis <u>F</u> ile	Default = 0
<u>C</u> entral <u>D</u> ifference <u>I</u> nterval	Default = $\epsilon^{4/15}$
<u>C</u> heck <u>F</u> requency	Default = 60
<u>C</u> rash <u>O</u> ption	Default = 3

<u>Crash Tolerance</u>	Default = 0.1
<u>Derivative Option</u>	Default = 1
<u>Defaults</u>	
<u>Derivative Linesearch</u>	Default
<u>Difference Interval</u>	Default = $\epsilon^{0.4}$
<u>Dump File</u>	Default = 0
<u>Elastic Mode</u>	Default = No
<u>Elastic Weight</u>	Default = 10^4
<u>Expand Frequency</u>	Default = 10000
<u>Factorization Frequency</u>	Default = 50
<u>Feasibility Tolerance</u>	Default = $1.0e - 6$
<u>Feasible Point</u>	
<u>Function Precision</u>	Default = $\epsilon^{0.8}$
<u>Hessian Full Memory</u>	Default = Full if $n_1 \leq 75$
<u>Hessian Limited Memory</u>	
<u>Hessian Frequency</u>	Default = 99999999
<u>Hessian Updates</u>	Default = 99999999
<u>Insert File</u>	Default = 0
<u>Infinite Bound Size</u>	Default = 10^{20}
<u>Linesearch Tolerance</u>	Default = 0.9
<u>List</u>	Default = Nolist
<u>Load File</u>	Default = 0
<u>LU Complete Pivoting</u>	
<u>LU Density Tolerance</u>	Default = 0.6
<u>LU Factor Tolerance</u>	Default = 3.99
<u>LU Partial Pivoting</u>	Default
<u>LU Rook Pivoting</u>	
<u>LU Singularity Tolerance</u>	Default = $\sqrt{\epsilon}$
<u>LU Update Tolerance</u>	Default = 3.00
<u>Major Feasibility Tolerance</u>	Default = $1.0e - 6$
<u>Major Iterations Limit</u>	Default = $\max\{1000, nf\}$
<u>Major Optimality Tolerance</u>	Default = $2.0e - 6$
<u>Major Print Level</u>	Default = 00001
<u>Major Step Limit</u>	Default = 2.0
<u>Maximize</u>	
<u>Minimize</u>	Default
<u>Minor Feasibility Tolerance</u>	Default = $1.0e - 6$
<u>Minor Iterations Limit</u>	Default = 500
<u>Minor Print Level</u>	Default = 1
<u>New Basis File</u>	Default = 0
<u>New Superbasics Limit</u>	Default = 99
<u>Nolist</u>	
<u>Nonderivative Linesearch</u>	
<u>Old Basis File</u>	Default = 0
<u>Partial Price</u>	Default = 1
<u>Pivot Tolerance</u>	Default = $10 \times \epsilon$
<u>Print File</u>	Default = 0
<u>Print Frequency</u>	Default = 100
<u>Proximal Point Method</u>	Default = 1
<u>Punch File</u>	Default = 0
<u>Save Frequency</u>	Default = 100
<u>Scale Option</u>	Default = 0
<u>Scale Tolerance</u>	Default = 0.9
<u>Solution File</u>	Default = 0
<u>Summary File</u>	Default = 0
<u>Summary Frequency</u>	Default = 100
<u>Superbasics Limit</u>	Default = $\min(500, n_1)$
<u>Suppress Parameters</u>	

<u>Timing Level</u>	Default = 0
<u>Unbounded Objective</u>	Default = 1.0e + 15
<u>Unbounded Step Size</u>	Default = 1.0e + 20
<u>Verify Level</u>	Default = 0
<u>Violation Limit</u>	Default = 1.0e + 6

11.2 Description of the Optional Arguments

Central Difference Interval – double r Default = $\epsilon^{4/15}$

When **Derivative Option** = 0, the central-difference interval r is used near an optimal solution to obtain more accurate (but more expensive) estimates of gradients. Twice as many function evaluations are required compared to forward differencing. The interval used for the j th variable $h_j = r(1 + |x_j|)$. The resulting derivative estimates should be accurate to $O(r^2)$, unless the functions are badly scaled.

Check Frequency – Integer i Default = 60

Every i th minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (the linear constraints and the linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Check Frequency = 1 is useful for debugging purposes, but otherwise this option should not be needed.

Crash Option – Integer i Default = 3

Crash Tolerance – double r Default = 0.1

Except on restarts, an internal Crash procedure is used to select an initial basis from certain rows and columns of the constraint matrix ($A -I$). The **Crash Option** i determines which rows and columns of A are eligible initially, and how many times the Crash procedure is called. Columns of $-I$ are used to pad the basis where necessary.

i	Meaning
0	The initial basis contains only slack variables: $B = I$.
1	The Crash procedure is called once, looking for a triangular basis in all rows and columns of the matrix A .
2	The Crash procedure is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and the Crash procedure is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
3	The Crash procedure is called up to three times (if there are nonlinear constraints). The first two calls treat <i>linear equalities</i> and <i>linear inequalities</i> separately. As before, the last call treats nonlinear rows before the first major iteration.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound). The Crash procedure then makes several passes through the columns of A , searching for a basis matrix that is essentially triangular. A column is assigned to 'pivot' on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

The **Crash Tolerance** r allows the starting Crash procedure to ignore certain 'small' non-zeros in each column of A . If a_{\max} is the largest element in column j , other non-zeros of a_{ij} in the columns are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by the Crash procedure may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

Defaults

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

Derivative Option – Integer i Default = 1

Derivative Option specifies which nonlinear function gradients are known analytically and will be supplied to `nag_opt_sparse_nlp_solve` (e04vhc) by you function **usrfun**.

i	Meaning
0	Some problem derivatives are unknown.
1	All problem derivatives are known.

The value $i = 1$ should be used whenever possible. It is the most reliable and will usually be the most efficient.

If $i = 0$, `nag_opt_sparse_nlp_solve` (e04vhc) will *estimate* the missing components of $G(x)$ using finite differences. This may simplify the coding of function **usrfun**. However, it could increase the total run-time substantially (since a special call to **usrfun** is required for each column of the Jacobian which has a missing element), and there is less assurance that an acceptable solution will be located. If the nonlinear variables are not well scaled, it may be necessary to specify a nonstandard **Difference Interval**.

For each column of the Jacobian, one call to **usrfun** is needed to estimate all missing elements in that column, if any. If the sparsity pattern of the Jacobian happens to be

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

where $*$ indicates known gradients and $?$ indicates unknown elements, `nag_opt_sparse_nlp_solve` (e04vhc) will use one call to **usrfun** to estimate the missing element in column 2, and another call to estimate both missing elements in column 3. No calls are needed for columns 1 and 4.

At times, central differences are used rather than forward differences. Twice as many calls to **usrfun** are needed. (This is not under your control.)

Derivative Linesearch Default
Nonderivative Linesearch

At each major iteration a line search is used to improve the merit function. A **Derivative Linesearch** uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step α_k . If some analytic derivatives are not provided, or a **Nonderivative Linesearch** is specified, `nag_opt_sparse_nlp_solve` (e04vhc) employs a line search based upon safeguarded quadratic interpolation, which does not require gradient evaluations.

A nonderivative line search can be slightly less robust on difficult problems, and it is recommended that the default be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative line search may give a significant decrease in computation time.

Difference Interval – double r Default = $\epsilon^{0.4}$

This alters the interval r that is used to estimate gradients by forward differences in the following circumstances:

- in the interval (‘cheap’) phase of verifying the problem derivatives;
- for verifying the problem derivatives;
- for estimating missing derivatives.

In all cases, a derivative with respect to x_j is estimated by perturbing that component of x to the value $x_j + r(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(r)$ unless the functions are badly scaled. Judicious alteration of r may sometimes lead to greater accuracy.

<u>Dump File</u> – Nag_FileID	i_1	Default = 0
<u>Load File</u> – Nag_FileID	i_2	Default = 0

(See Section 11.1 for a description of **Nag_FileID**.)

Dump File and **Load File** are similar to **Punch File** and **Insert File**, but they record solution information in a manner that is more direct and more easily modified. A full description of information recorded in **Dump File** and **Load File** is given in Gill *et al.* (1999).

If **Dump File** > 0, the last solution obtained will be output to the file associated with ID i_1 .

If **Load File** > 0, the file associated with ID i_2 , containing basis information, will be read. The file will usually have been output previously as a **Dump File**. The file will not be accessed if an **Old Basis File** or an **Insert File** is specified.

<u>Elastic Mode</u>		Default = No
----------------------------	--	--------------

Normally nag_opt_sparse_nlp_solve (e04vhc) initiates elastic mode only when it seems necessary. Setting **Elastic Mode** = Yes causes elastic mode to be entered from the beginning.

<u>Elastic Weight</u> – double	r	Default = 10^4
---------------------------------------	-----	------------------

This keyword determines the initial weight γ associated with the problem (12) (see Section 10.5).

At major iteration k , if elastic mode has not yet started, a scale factor $\sigma_k = 1 + \|g(x_k)\|_\infty$ is defined from the current objective gradient. Elastic mode is then started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\sigma_k r$. The QP is resolved in elastic mode with $\gamma = \sigma_k r$.

Thereafter, major iterations continue in elastic mode until they converge to a point that is optimal for (12) (see Section 10.5). If the point is feasible for ($v = w = 0$), it is declared locally optimal. Otherwise, γ is increased by a factor of 10 and major iterations continue.

<u>Expand Frequency</u> – Integer	i	Default = 10000
--	-----	-----------------

This option is part of the anti-cycling procedure designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the **Minor Feasibility Tolerance** is δ . Over a period of i iterations, the tolerance actually used by nag_opt_sparse_nlp_solve (e04vhc) increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing i helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see **Pivot Tolerance**).

<u>Factorization Frequency</u> – Integer	i	Default = 50
---	-----	--------------

At most i basis changes will occur between factorizations of the basis matrix.

With linear programs, the basis factors are usually updated every iteration. The default i is reasonable for typical problems. Higher values up to $i = 100$ (say) may be more efficient on well scaled problems.

When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the **Check Frequency**) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of i updates is reached.

<u>Feasibility Tolerance</u> – Integer	i	Default = $1.0e - 6$
---	-----	----------------------

See **Minor Feasibility Tolerance**.

Function Precision – double r Default = $\epsilon^{0.8}$

The *relative function precision* ϵ_r is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_r would be $1.0e - 6$.

Ideally the functions $f_i(x)$ should have magnitude of order 1. If all functions are substantially *less* than 1 in magnitude, ϵ_r should be the *absolute* precision. For example, if $f(x) = 1.23456789e - 4$ at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_r would be $1.0e - 10$.)

The default value of ϵ_r is appropriate for simple analytic functions.

In some cases the function values will be the result of extensive computation, possibly involving a costly iterative procedure that can provide few digits of precision. Specifying an appropriate **Function Precision** may lead to savings, by allowing the line search procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Hessian Full Memory Default = Full if $n_1 \leq 75$
Hessian Limited Memory

These options select the method for storing and updating the approximate Hessian. (nag_opt_sparse_nlp_solve (e04vhc) uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If **Hessian Full Memory** is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect 1-step Q-superlinear convergence to the solution.

Hessian Limited Memory should be used on problems where n_1 is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

Hessian Frequency – Integer i Default = 99999999

If **Hessian Full Memory** is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

Hessian Full Memory and **Hessian Frequency** = 20 have a similar effect to **Hessian Limited Memory** and **Hessian Updates** = 20 (except that the latter retains the current diagonal during resets).

Hessian Updates – Integer i Default = 99999999

If **Hessian Limited Memory** is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Infinite Bound Size – double 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 plus infinity (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as minus infinity). If $r \leq 0$, the default value is used.

Linesearch Tolerance – double r Default = 0.9

This tolerance, r , controls the accuracy with which a steplength will be located along the direction of each search iteration. At the start of each line search a target directional derivative for the merit function is identified. This argument determines the accuracy to which this target value is approximated.

r must be a double value in the range $0.0 \leq r \leq 1.0$.

The default value $r = 0.9$ requests just moderate accuracy in the line search.

If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try $r = 0.1, 0.01$ or 0.001 .

If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. *If all gradients are known*, try $r = 0.99$. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)

If not all gradients are known, a moderately accurate search remains appropriate. Each search will require only 1–5 function values (typically), but many function calls will then be needed to estimate missing gradients for the next iteration.

List
Nolist

Default = **Nolist**

For `nag_opt_sparse_nlp_solve` (e04vhc), normally each optional argument specification is printed as it is supplied. **Nolist** may be used to suppress the printing and **List** may be used to turn on printing.

LU Factor Tolerance – double	r_1	Default = 3.99
LU Update Tolerance – double	r_2	Default = 3.00

The values of r_1 and r_2 affect the stability of the basis factorization $B = LU$, during refactorization and updates respectively. The lower triangular matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix}$$

where the multipliers μ will satisfy $|\mu| \leq r_i$. The default values of r_1 and r_2 usually strike a good compromise between stability and sparsity. They must satisfy $r_1, r_2 \geq 1.0$.

For large and relatively dense problems, $r_1 = 10.0$ or 5.0 (say) may give a useful improvement in stability without impairing sparsity to a serious degree.

LU Partial Pivoting

Default

LU Rook Pivoting

LU Complete Pivoting

The LU factorization implements a Markowitz-type search for a pivot that locally minimizes the fill-in subject to a threshold pivoting stability criterion. The default option is to use threshold partial pivoting. The options **LU Rook Pivoting** and **LU Complete Pivoting** are more expensive than partial pivoting but are more stable and better at revealing rank.

LU Density Tolerance – double	r_1	Default = 0.6
LU Singularity Tolerance – double	r_2	Default = $\sqrt{\epsilon}$

The density tolerance, r_1 , is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1 , the Markowitz strategy for choosing pivots is terminated. The remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

The singularity tolerance, r_2 , helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the j th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)

In some cases, the Jacobian matrix may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting a larger tolerance $r_2 = 1.0e - 5$, say, may help cause a judicious change of basis.

Major Feasibility Tolerance – double r Default = 1.0e – 6

This tolerance, r , specifies how accurately the nonlinear constraints should be satisfied. The default value is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let $rowerr$ be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$rowerr = \max_i viol_i / \|x\| \leq r, \quad (13)$$

where $viol_i$ is the violation of the i th nonlinear constraint, for $i = 1, 2, \dots, \mathbf{nf}$.

In the major iteration log (see Section), $rowerr$ appears as the quantity labelled ‘Feasible’. If some of the problem functions are known to be of low accuracy, a larger **Major Feasibility Tolerance** may be appropriate.

Major Optimality Tolerance – double r Default = 2.0e – 6

This tolerance, r , specifies the final accuracy of the dual variables. On successful termination, `nag_opt_sparse_nlp_solve` (e04vhc) will have computed a solution (x, s, π) such that

$$maxComp = \max_j Comp_j / \|\pi\| \leq r, \quad (14)$$

where $Comp_j$ is an estimate of the complementarity slackness for variable j , for $j = 1, 2, \dots, \mathbf{n} + \mathbf{nf}$. The values $Comp_j$ are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the j th component of the objective gradient, a_j is the associated column of the constraint matrix $(A \ -I)$, and π is the set of QP dual variables):

$$Comp_j = \begin{cases} d_j \min \{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min \{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases} \quad (15)$$

In the **Print File**, $maxComp$ appears as the quantity labelled ‘Optimal’.

Major Iterations Limit – Integer i Default = $\max \{1000, \mathbf{nf}\}$

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints.

Major Print Level – Integer i Default = 00001

This controls the amount of output to the **Print File** and **Summary File** at each major iteration. **Major Print Level** = 0 suppresses most output, except for error messages. **Major Print Level** = 1 gives normal output for linear and nonlinear problems, and **Major Print Level** = 11 gives additional details of the Jacobian factorization that commences each major iteration.

In general, the value being specified may be thought of as a binary number of the form

Major Print Level *JFDXbs*

where each letter stands for a digit that is either 0 or 1 as follows:

- s* a single line that gives a summary of each major iteration. (This entry in *JFDXbs* is not strictly binary since the summary line is printed whenever $JFDXbs \geq 1$);
- b* basis statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if $JFDXbs \geq 10$);
- X* x_k , the nonlinear variables involved in the objective function or the constraints;
- D* π_k , the dual variables for the nonlinear constraints;
- F* $f_0(x_k)$, the values of the nonlinear constraint functions;
- J* $J(x_k)$, the Jacobian matrix.

To obtain output of any items *JFDXbs*, set the corresponding digit to 1, otherwise to 0.

If $J = 1$, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if $J = 1$, there is no reason to specify $X = 1$ unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3 1.250000e+01 BS 1 1.00000e+00 4 2.00000e+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

Major Step Limit – double r Default = 2.0

This argument limits the change in x during a line search. It applies to all nonlinear problems, once a ‘feasible solution’ or ‘feasible subproblem’ has been found.

1. A line search determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried as $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The argument r is therefore used to define a limit $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The **Major Step Limit** provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A ‘good’ starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Minimize Default
Maximize
Feasible Point

The keywords **Minimize** and **Maximize** specify the required direction of optimization. It applies to both linear and nonlinear terms in the objective.

The keyword **Feasible Point** means ‘Ignore the objective function’ while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible without altering the call to `nag_opt_sparse_nlp_solve` (e04vhc).

Minor Feasibility Tolerance – double r Default = $1.0e - 6$

`nag_opt_sparse_nlp_solve` (e04vhc) tries to ensure that all variables eventually satisfy their upper and lower bounds to within this tolerance, r . This includes slack variables. Hence, general linear constraints should also be satisfied to within r .

Feasibility with respect to nonlinear constraints is judged by the **Major Feasibility Tolerance** (not by r).

If the bounds and linear constraints cannot be satisfied to within r , the problem is declared *infeasible*. Let `sInf` be the corresponding sum of infeasibilities. If `sInf` is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem.

For example, if $f(x) = \sqrt{x_1} + \log(x_2)$, it is essential to place lower bounds on both variables. If $r = 1.0e - 6$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)

If **Scale Option** ≥ 1 , feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).

In reality, `nag_opt_sparse_nlp_solve` (e04vhc) uses r as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. `nag_opt_sparse_nlp_solve` (e04vhc) is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the **Elastic Mode** option.

Minor Iterations Limit – Integer i Default = 500

If the number of minor iterations for the optimality phase of the QP subproblem exceeds i , then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality.

Note that more than i minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the line search.

In the major iteration log (see Section) a ‘t’ at the end of a line indicates that the corresponding QP was artificially terminated using the limit i .

Note that **Minor Iterations Limit** defines an independent *absolute* limit on the *total* number of minor iterations (summed over all QP subproblems).

Minor Print Level – Integer i Default = 1

This controls the amount of output to the **Print File** and **Summary File** during solution of the QP subproblems. The value of i has the following effect:

i	Meaning
0	No minor iteration output except error messages.
≥ 1	A single line of output at each minor iteration (controlled by Print Frequency and Summary Frequency).
≥ 10	Basis factorization statistics generated during the periodic refactorization of the basis (see Factorization Frequency). Statistics for the <i>first factorization</i> each major iteration are controlled by the Major Print Level .

New Basis File – Nag_FileID i_1 Default = 0
Backup Basis File – Nag_FileID i_2 Default = 0
Save Frequency – Integer i_3 Default = 100

(See Section 11.1 for a description of **Nag_FileID**.)

New Basis File and **Backup Basis File** are sometimes referred to as basis maps. They contain the most compact representation of the state of each variable. They are intended for restarting the solution of a problem at a point that was reached by an earlier run. For non-trivial problems, it is advisable to save basis maps at the end of a run, in order to restart the run if necessary.

If **New Basis File** > 0 , a basis map will be saved in the file associated with ID i_1 every i_3 th iteration. The first record of the file will contain the word PROCEEDING if the run is still in progress. A basis map will also be saved at the end of a run, with some other word indicating the final solution status.

If **Backup Basis File** > 0 , it is intended as a safeguard against losing the results of a long run. Suppose that a **New Basis File** is being saved every 100 (**Save Frequency**) iterations, and that `nag_opt_sparse_nlp_solve` (e04vhc) is about to save such a basis at iteration 2000. It is conceivable that the run may be interrupted during the next few milliseconds (in the middle of the save). In this case the basis file will be corrupted and the run will have been essentially wasted.

To eliminate this risk, both a **New Basis File** and a **Backup Basis File** may be specified using calls of `nag_open_file` (x04acc).

The current basis will then be saved every 100 iterations, first in the **New Basis File** and then immediately in the **Backup Basis File**. If the run is interrupted at iteration 2000 during the save in the **New Basis File** there will still be a usable basis in the **Backup Basis File**.

Note that a new basis will be saved in **New Basis File** at the end of a run if it terminates normally, but it will not be saved in **Backup Basis File**. In the above example, if an optimum solution is found at iteration

2050 (or if the iteration limit is 2050), the final basis in the **Backup Basis File** will correspond to iteration 2050, but the last basis saved in the **New Basis File** will be the one for iteration 2000.

A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (1999).

New Superbasics Limit – Integer i Default = 99

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than i the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality.

In the major iteration log (see Section), a ‘T’ at the end of a line indicates that the QP was terminated early in this way.

Old Basis File – Nag_FileID i Default = 0

(See Section 11.1 for a description of **Nag_FileID**.)

If **Old Basis File** > 0 , the basis maps information will be obtained from the file associated with ID i . A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (1999). The file will usually have been output previously as a **New Basis File** or **Backup Basis File**.

The file will not be acceptable if the number of rows or columns in the problem has been altered.

Partial Price – Integer i Default = 1

This argument is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each ‘pricing’ operation (when a nonbasic variable is selected to become superbasic).

When $i = 1$, all columns of the constraint matrix ($A \ -I$) are searched.

Otherwise, A and I are partitioned to give i roughly equal segments A_j, I_j , for $j = 1, \dots, i$. If the previous pricing search was successful on A_j, I_j , the next search begins on the segments A_{j+1}, I_{j+1} . (All subscripts here are modulo i .)

If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2} , and so on.

Partial Price r (or $r/2$ or $r/3$) may be appropriate for time-stage models having r time periods.

Pivot Tolerance – double r Default = $10 \times \epsilon$

During the solution of QP subproblems, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

When x changes to $x + \alpha p$ for some search direction p , a ‘ratio test’ is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.

Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r .

It is common for two or more variables to reach a bound at essentially the same time. In such cases, the **Minor Feasibility Tolerance** (say, t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.

To a lesser extent, the **Expand Frequency** (say, f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

Print File – Nag_FileID i Default = 0

(See Section 11.1 for a description of **Nag_FileID**.)

If **Print File** > 0 , the following information is output to a file associated with ID i during the solution of each problem:

- a listing of the optional arguments;
- some statistics about the problem;
- the amount of storage available for the *LU* factorization of the basis matrix;
- notes about the initial basis resulting from a Crash procedure or a basis file;
- the iteration log;
- basis factorization statistics;
- the exit **fail** condition and some statistics about the solution obtained;
- the printed solution, if requested.

These items are described in Sections 8 and 12. Further brief output may be directed to the **Summary File**.

Print Frequency – Integer *i* Default = 100

If $i > 0$, one line of the iteration log will be printed every i th iteration. A value such as $i = 10$ is suggested for those interested only in the final solution.

Proximal Point Method – Integer *i* Default = 1

$i = 1$ or 2 specifies minimization of $\|x - x_0\|_1$ or $\frac{1}{2}\|x - x_0\|_2^2$ when the starting point x_0 is changed to satisfy the linear constraints (where x_0 refers to nonlinear variables).

Punch File – Nag_FileID *i*₁ Default = 0

Insert File – Nag_FileID *i*₂ Default = 0

(See Section 11.1 for a description of **Nag_FileID**.)

The **Punch File** from a previous run may be used as an **Insert File** for a later run on the same problem. A full description of information recorded in **Insert File** and **Punch File** is given in Gill *et al.* (1999).

If **Punch File** > 0 , the final solution obtained will be output to the file associated with ID i_1 . For linear programs, this format is compatible with various commercial systems.

If **Insert File** > 0 , the file associated with ID i_2 , containing basis information, will be read. The file will usually have been output previously as a **Punch File**. The file will not be accessed if **Old Basis File** is specified.

Scale Option – Integer *i* Default = 0

Scale Tolerance – double *r* Default = 0.9

Three scale options are available as follows:

<i>i</i>	Meaning
0	No scaling. This is recommended if it is known that x and the constraint matrix never have very large elements (say, larger than 1000).
1	The constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer (1982)). This will sometimes improve the performance of the solution procedures.
2	The constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of $(A \ -I)$ that are fixed or have positive lower bounds or negative upper bounds.

Scale Tolerance affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest non-zero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \quad (a_{ij} \neq 0).$$

If $\max \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and

column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A . At most 10 passes are made.

Solution File – Nag_FileID i Default = 0

(See Section 11.1 for a description of **Nag_FileID**.)

If **Solution File** > 0, the final solution will be output to the file associated with ID i .

To see more significant digits in the printed solution, it will sometimes be useful to specify that the **Solution File** refers to the **Print File**.

Summary File – Nag_FileID i_1 Default = 0

Summary Frequency – Integer i_2 Default = 100

(See Section 11.1 for a description of **Nag_FileID**.)

If **Summary File** > 0, a brief log will be output to the file associated with i_1 , including one line of information every i_2 th iteration. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. (If something looks wrong, the run can be manually terminated.) Further details are given in Section 12.6.

Superbasics Limit – Integer i Default = $\min(500, n_1)$

This option places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the ‘number of degrees of freedom’ expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than m , the number of general constraints.) The default value of i is therefore 1.

For nonlinear problems, the number of degrees of freedom is often called the ‘number of independent variables’.

Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables.

For many problems, i may be considerably smaller than n_1 . This will save storage if n_1 is very large.

Suppress Parameters

Normally `nag_opt_sparse_nlp_solve` (e04vhc) prints the options file as it is being read, and then prints a complete list of the available keywords and their final values. The **Suppress Parameters** option tells `nag_opt_sparse_nlp_solve` (e04vhc) not to print the full list.

Timing Level – Integer i Default = 0

If $i > 0$, some timing information will be output to the **Print File**.

Unbounded Objective – double r_1 Default = $1.0e + 15$

Unbounded Step Size – double r_2 Default = $1.0e + 20$

These arguments are intended to detect unboundedness in nonlinear problems. During a line search, F is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. If $|F|$ exceeds r_1 or α exceeds r_2 , iterations are terminated with the exit message

```
Problem is unbounded (or badly scaled)
```

If singularities are present, unboundedness in $f_0(x)$ may be manifested by a floating-point overflow (during the evaluation of $f_0(x + \alpha p)$), before the test against r_1 can be made.

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

Verify Level – Integer i Default = 0

This option refers to finite-difference checks on the derivatives computed by the user-supplied functions. Derivatives are checked at the first point that satisfies all bounds and linear constraints.

<i>i</i>	Meaning
0	Only a ‘cheap’ test will be performed, requiring two calls to usrfun .
1	Individual gradients will be checked (with a more reliable test). A key of the form OK or Bad? indicates whether or not each component appears to be correct.
2	Individual columns of the problem Jacobian will be checked.
3	Options 2 and 1 will both occur (in that order).
-1	Derivative checking is disabled.

Verify Level = 3 should be specified whenever a new user-supplied function **usrfun** is being developed.

Violation Limit – double *r* Default = 1.0e + 6

This keyword defines an absolute limit on the magnitude of the maximum constraint violation, r , after the line search. On completion of the line search, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq r \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the i th nonlinear constraint violation $v_i(x) = \max(0, l_i - f(x), f(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of r . This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then r may be any large positive value.

12 Description of Monitoring Information

nag_opt_sparse_nlp_solve (e04vhc) produces monitoring information, statistical information and information about the solution. Section 8.1 contains details of the final output information sent to **Print File**. This section contains other details of output information.

12.1 Major Iteration Log

This section describes the output to **Print File** if **Major Print Level** > 0. One line of information is output every k th major iteration, where k is **Print Frequency**.

Label	Description
Itns	is the cumulative number of minor iterations.
Major	is the current major iteration number.
Minors	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, Minors will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 10).
Step	is the step length α taken along the current search direction p . The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.
nCon	the number of times function usrfun has been called to evaluate the nonlinear problem functions. Evaluations needed for the estimation of the derivatives by finite differences are not included. nCon is printed as a guide to the amount of work required for the line search.
Feasible	is the value of <i>rowerr</i> (see (13)), the maximum component of the scaled nonlinear constraint residual (see Major Feasibility Tolerance). The solution is regarded as acceptably feasible if Feasible is less than the Major Feasibility Tolerance . In this case, the entry is contained in parentheses. If the constraints are linear, all iterates are feasible and this entry is not printed.
Optimal	is the value of <i>maxComp</i> (see (14)), the maximum complementary gap (see Major Optimality Tolerance). It is an estimate of the degree of nonoptimality of the

	reduced costs. Both <code>Feasible</code> and <code>Optimal</code> are small in the neighbourhood of a solution.
<code>MeritFunction</code>	<p>is the value of the augmented Lagrangian merit function (see (8)). This function will decrease at each iteration unless it was necessary to increase the penalty arguments (see Section 10.4). As the solution is approached, <code>MeritFunction</code> will converge to the value of the objective at the solution.</p> <p>In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.</p> <p>If the constraints are linear, this item is labelled <code>Objective</code>, the value of the objective function. It will decrease monotonically to its optimal value.</p>
<code>L+U</code>	<p>is the number of non-zeros representing the basis factors L and U on completion of the QP subproblem.</p> <p>If nonlinear constraints are present, the basis factorization $B = LU$ is computed at the start of the first minor iteration. At this stage, $LU = \text{lenL} + \text{lenU}$, where lenL (see Section 12.3) is the number of subdiagonal elements in the columns of a lower triangular matrix and lenU (see Section 12.3) is the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix.</p> <p>As columns of B are replaced during the minor iterations, LU may fluctuate up or down but, in general, will tend to increase. As the solution is approached and the minor iterations decrease towards zero, LU will reflect the number of non-zeros in the LU factors at the start of the QP subproblem.</p> <p>If the constraints are linear, refactorization is subject only to the Factorization Frequency, and LU will tend to increase between factorizations.</p>
<code>BSwap</code>	is the number of columns of the basis matrix B that were swapped with columns of S to improve the condition of B . The swaps are determined by an LU factorization of the rectangular matrix $B_S = (B S)^T$ with stability being favoured more than sparsity.
<code>nS</code>	is the current number of superbasic variables.
<code>CondHz</code>	<p>is an estimate of the condition number of $R^T R$, an estimate of $Z^T H Z$, the reduced Hessian of the Lagrangian. It is the square of the ratio of the largest and smallest diagonals of the upper triangular matrix R (which is a lower bound on the condition number of $R^T R$). <code>CondHz</code> gives a rough indication of whether or not the optimization procedure is having difficulty. If ϵ is the relative machine precision being used, the SQP algorithm will make slow progress if <code>CondHz</code> becomes as large as $\epsilon^{-1/2} \approx 10^8$, and will probably fail to find a better solution if <code>CondHz</code> reaches $\epsilon^{-3/4} \approx 10^{12}$.</p> <p>To guard against high values of <code>CondHz</code>, attention should be given to the scaling of the variables and the constraints. In some cases it may be necessary to add upper or lower bounds to certain variables to keep them a reasonable distance from singularities in the nonlinear functions or their derivatives.</p>
<code>Penalty</code>	is the Euclidean norm of the vector of penalty arguments used in the augmented Lagrangian merit function (not printed if there are no nonlinear constraints).

The summary line may include additional code characters that indicate what happened during the course of the major iteration.

Label	Description
<code>c</code>	central differences have been used to compute the unknown components of the objective and constraint gradients. A switch to central differences is made if either the line search gives a small step, or x is close to being optimal. In some cases, it may be necessary to re-solve the QP subproblem with the central difference gradient and Jacobian.

d	during the line search it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of Violation Limit .
l	the norm wise change in the variables was limited by the value of the Major Step Limit . If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of Major Step Limit .
i	if nag_opt_sparse_nlp_solve (e04vhc) is not in elastic mode, an ‘i’ signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem (12) (see Section 10.5). If nag_opt_sparse_nlp_solve (e04vhc) is already in elastic mode, an ‘i’ indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)
M	an extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.
m	this is the same as M except that it was also necessary to modify the update to include an augmented Lagrangian term.
n	no positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.
R	the approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the Hessian Frequency and Hessian Updates keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.
r	the approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.
s	a self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.
t	the minor iterations were terminated because of the Minor Iterations Limit .
T	the minor iterations were terminated because of the New Superbasics Limit .
u	the QP subproblem was unbounded.
w	a weak solution of the QP subproblem was found.
z	the Superbasics Limit was reached.

12.2 Minor Iteration Log

If **Minor Print Level** > 0, one line of information is output to the **Print File** every k th minor iteration, where k is the specified **Print Frequency**. A heading is printed before the first such line following a basis factorization. The heading contains the items described below. In this description, a pricing operation is defined to be the process by which a nonbasic variable is selected to become superbasic (in addition to those already in the superbasic set). The selected variable is denoted by jq . Variable jq often becomes basic immediately. Otherwise it remains superbasic, unless it reaches its opposite bound and returns to the nonbasic set.

If **Partial Price** is in effect, variable jq is selected from A_{pp} or I_{pp} , the p th segments of the constraint matrix $(A \ -I)$.

Label	Description
Itn	the current iteration number.

RedCost or QPmult	is the reduced cost (or reduced gradient) of the variable <code>jq</code> selected by the pricing procedure at the start of the present iteration. Algebraically, dg is $d_j = g_j - \pi^T a_j$ for $j = jq$, where g_j is the gradient of the current objective function, π is the vector of dual variables for the QP subproblem, and a_j is the j th column of $(A \ -I)$. Note that d_j is the 1-norm of the reduced-gradient vector at the start of the iteration, just after the pricing procedure.
LPstep or QPstep	is the step length α taken along the current search direction p . The variables x have just been changed to $x + \alpha p$. If a variable is made superbasic during the current iteration (<code>+SBS</code> > 0), <code>Step</code> will be the step to the nearest bound. During Phase 2, the step can be greater than one only if the reduced Hessian is not positive-definite.
nInf	is the number of infeasibilities <i>after</i> the present iteration. This number will not increase unless the iterations are in elastic mode.
SumInf	If <code>nInf</code> > 0, this is <code>sInf</code> , the sum of infeasibilities after the present iteration. It usually decreases at each non-zero <code>Step</code> , but if <code>nInf</code> decreases by 2 or more, <code>SumInf</code> may occasionally increase. In elastic mode, the heading is changed to <code>Composite Obj</code> , and the value printed decreases monotonically.
rgNorm	is the norm of the reduced-gradient vector at the start of the iteration. (It is the norm of the vector with elements d_j for variables j in the superbasic set.) During Phase 2 this norm will be approximately zero after a unit step. (The heading is not printed if the problem is linear.)
LPobjective or QPobjective	the QP objective function after the present iteration. In elastic mode, the heading is changed to <code>Elastic QPobj</code> . In either case, the value printed decreases monotonically.
+SBS	is the variable <code>jq</code> selected by the pricing operation to be added to the superbasic set.
-SBS	is the variable chosen to leave the set of superbasics. It has become basic if the entry under <code>-B</code> is non-zero; otherwise it has become nonbasic.
-BS	is the variable removed from the basis (if any) to become nonbasic.
-B	is the variable removed from the basis (if any) to swap with a slack variable made superbasic by the latest pricing operation. The swap is done to ensure that there are no superbasic slacks.
Pivot	if column a_q replaces the r th column of the basis B , <code>Pivot</code> is the r th element of a vector y satisfying $By = a_q$. Wherever possible, <code>Step</code> is chosen to avoid extremely small values of <code>Pivot</code> (since they cause the basis to be nearly singular). In rare cases, it may be necessary to increase the Pivot Tolerance to exclude very small elements of y from consideration during the computation of <code>Step</code> .
L+U	is the number of non-zeros representing the basis factors L and U . Immediately after a basis factorization $B = LU$, this is <code>lenL+lenU</code> , the number of subdiagonal elements in the columns of a lower triangular matrix and the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. Further non-zeros are added to L when various columns of B are later replaced. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of L will steadily increase, whereas the value of U may fluctuate up or down. Thus the value of $L+U$ may fluctuate up or down (in general, it will tend to increase).
ncp	is the number of compressions required to recover storage in the data structure for U . This includes the number of compressions needed during the previous basis factorization.

nS is the current number of superbasic variables. (The heading is not printed if the problem is linear.)

CondHz see the major iteration log. (The heading is not printed if the problem is linear.)

12.3 Basis Factorization Statistics

If **Major Print Level** ≥ 10 , the following items are output to the **Print File** whenever the basis B or the rectangular matrix $B_S = (B S)^T$ is factorized before solution of the next QP subproblem (see Section 11.2).

Note that B_S may be factorized at the start of just some of the major iterations. It is immediately followed by a factorization of B itself.

Gaussian elimination is used to compute a sparse LU factorization of B or B_S , where PLP^T and PUQ are lower and upper triangular matrices for some permutation matrices P and Q . Stability is ensured as described under **LU Factor Tolerance** (see Section 11.2).

If **Minor Print Level** ≥ 10 (see Section 11.2), the same items are printed during the QP solution whenever the current B is factorized.

Label	Description														
Factorize	the number of factorizations since the start of the run.														
Demand	a code giving the reason for the present factorization.														
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;">Code</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>First LU factorization.</td> </tr> <tr> <td>1</td> <td>The number of updates reached the Factorization Frequency (see Section 11.2).</td> </tr> <tr> <td>2</td> <td>The non-zeros in the updated factors have increased significantly.</td> </tr> <tr> <td>7</td> <td>Not enough storage to update factors.</td> </tr> <tr> <td>10</td> <td>Row residuals are too large (see the description of Check Frequency in Section 11.2).</td> </tr> <tr> <td>11</td> <td>Ill-conditioning has caused inconsistent results.</td> </tr> </tbody> </table>	Code	Meaning	0	First LU factorization.	1	The number of updates reached the Factorization Frequency (see Section 11.2).	2	The non-zeros in the updated factors have increased significantly.	7	Not enough storage to update factors.	10	Row residuals are too large (see the description of Check Frequency in Section 11.2).	11	Ill-conditioning has caused inconsistent results.
Code	Meaning														
0	First LU factorization.														
1	The number of updates reached the Factorization Frequency (see Section 11.2).														
2	The non-zeros in the updated factors have increased significantly.														
7	Not enough storage to update factors.														
10	Row residuals are too large (see the description of Check Frequency in Section 11.2).														
11	Ill-conditioning has caused inconsistent results.														
Itn	is the current minor iteration number.														
Nonlin	is the number of nonlinear variables in the current basis B .														
Linear	is the number of linear variables in B .														
Slacks	is the number of slack variables in B .														
B, BR, BS or BT factorize	is the type of LU factorization.														
	<table border="0" style="width: 100%;"> <tbody> <tr> <td>B</td> <td>periodic factorization of the basis B.</td> </tr> <tr> <td>BR</td> <td>more careful rank-revealing factorization of B using threshold rook pivoting. This occurs mainly at the start, if the first basis factors seem singular or ill-conditioned. Followed by a normal B factorize.</td> </tr> <tr> <td>BS</td> <td>B_S is factorized to choose a well-conditioned B from the current $(B S)$. Followed by a normal B factorize.</td> </tr> <tr> <td>BT</td> <td>same as BS except the current B is tried first and accepted if it appears to be not much more ill-conditioned than after the previous BS factorize.</td> </tr> </tbody> </table>	B	periodic factorization of the basis B .	BR	more careful rank-revealing factorization of B using threshold rook pivoting. This occurs mainly at the start, if the first basis factors seem singular or ill-conditioned. Followed by a normal B factorize.	BS	B_S is factorized to choose a well-conditioned B from the current $(B S)$. Followed by a normal B factorize.	BT	same as BS except the current B is tried first and accepted if it appears to be not much more ill-conditioned than after the previous BS factorize.						
B	periodic factorization of the basis B .														
BR	more careful rank-revealing factorization of B using threshold rook pivoting. This occurs mainly at the start, if the first basis factors seem singular or ill-conditioned. Followed by a normal B factorize.														
BS	B_S is factorized to choose a well-conditioned B from the current $(B S)$. Followed by a normal B factorize.														
BT	same as BS except the current B is tried first and accepted if it appears to be not much more ill-conditioned than after the previous BS factorize.														
m	is the number of rows in B or B_S .														
n	is the number of columns in B or B_S . Preceded by '=' or '>' respectively.														
Elms	is the number of non-zero elements in B or B_S .														
Amax	is the largest non-zero in B or B_S .														
Density	is the percentage non-zero density of B or B_S .														
Merit	is the average Markowitz merit count for the elements chosen to be the diagonals of PUQ . Each merit count is defined to be $(c - 1)(r - 1)$ where c and r are the														

	number of non-zeros in the column and row containing the element at the time it is selected to be the next diagonal. <i>Merit</i> is the average of <i>n</i> such quantities. It gives an indication of how much work was required to preserve sparsity during the factorization.
lenL	is the number of non-zeros in <i>L</i> .
Cmpressns	is the number of times the data structure holding the partially factored matrix needed to be compressed to recover unused storage. Ideally this number should be zero. If it is more than 3 or 4, the amount of workspace available to <code>nag_opt_sparse_nlp_solve</code> (e04vhc) should be increased for efficiency.
Inces	is the percentage increase in the number of non-zeros in <i>L</i> and <i>U</i> relative to the number of non-zeros in <i>B</i> or <i>B_S</i> .
Utri	is the number of triangular rows of <i>B</i> or <i>B_S</i> at the top of <i>U</i> .
lenU	the number of non-zeros in <i>U</i> .
Ltol	is the maximum subdiagonal element allowed in <i>L</i> . This is the specified LU Factor Tolerance (see Section 11.2) or a smaller value that is currently being used for greater stability.
Umax	the maximum non-zero element in <i>U</i> .
Ugrwth	is the ratio U_{\max}/A_{\max} , which ideally should not be substantially larger than 10.0 or 100.0. If it is orders of magnitude larger, it may be advisable to reduce the LU Factor Tolerance to 5.0, 4.0, 3.0 or 2.0, say (but bigger than 1.0) (see Section 11.2). As long as L_{\max} is not large (say 10.0 or less), $\max\{A_{\max}, U_{\max}\}/DU_{\min}$ gives an estimate of the condition number <i>B</i> . If this is extremely large, the basis is nearly singular. Slacks are used to replace suspect columns of <i>B</i> and the modified basis is refactored.
Ltri	is the number of triangular columns of <i>B</i> or <i>B_S</i> at the left of <i>L</i> .
dense1	is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.
Lmax	is the actual maximum subdiagonal element in <i>L</i> (bounded by <i>Ltol</i>).
Akmax	is the largest non-zero generated at any stage of the <i>LU</i> factorization. (Values much larger than A_{\max} indicate instability.)
growth	is the ratio A_{\max}/A_{\min} . Values much larger than 100 (say) indicate instability.
bump	is the size of the ‘bump’ or block to be factorized nontrivially after the triangular rows and columns of <i>B</i> or <i>B_S</i> have been removed.
dense2	is the number of columns remaining when the density of the basis matrix being factorized reached 0.6. (The Markowitz pivot strategy searches fewer columns at that stage.)
DUmax	is the largest diagonal of <i>PUQ</i> .
DUmin	is the smallest diagonal of <i>PUQ</i> .
condU	the ratio DU_{\max}/DU_{\min} , which estimates the condition number of <i>U</i> (and of <i>B</i> if <i>Ltol</i> is less than 100, say).

12.4 Crash Statistics

If **Major Print Level** ≥ 10 , the following items are output to the **Print File** when **start** = **Nag_Cold** and no basis file is loaded (see Section 11.2). They refer to the number of columns that the Crash procedure selects during selected passes through *A* while searching for a triangular basis matrix.

Label	Description
Slacks	is the number of slacks selected initially.
Free cols	is the number of free columns in the basis, including those whose bounds are rather far apart.
Preferred	is the number of ‘preferred’ columns in the basis (i.e., $hs(j) = 3$ for some $j \leq n$). It will be a subset of the columns for which $hs(j) = 3$ was specified.
Unit	is the number of unit columns in the basis.
Double	is the number of columns in the basis containing 2 non-zeros.
Triangle	is the number of triangular columns in the basis with 3 or more non-zeros.
Pad	is the number of slacks used to pad the basis (to make it a nonsingular triangle).

12.5 The Solution File

At the end of a run, the final solution may be output as a solution file, according to **Solution File** (see Section 11.2). Some header information appears first to identify the problem and the final state of the optimization procedure. A ROWS section and a COLUMNS section then follow, giving one line of information for each row and column. The format used is similar to certain commercial systems, though there is no industry standard.

The maximum record length is 111 characters.

To reduce clutter, a full stop (.) is printed for any numerical value that is exactly zero. The values ± 1 are also printed specially as 1.0 and -1.0 . Infinite bounds ($\pm 10^{20}$ or larger) are printed as None.

A solution file is intended to be read from disk by a self-contained program that extracts and saves certain values as required for possible further computation. Typically, the first 14 records would be ignored. The end of the ROWS section is marked by a record that starts with a 1 and is otherwise blank. If this and the next 4 records are skipped, the COLUMNS section can then be read under the same format.

A full description of the ROWS section and the COLUMNS section is given in Section 8.1.

12.6 The Summary File

If **Summary File** > 0 , the following information is output to the **Summary File**. (It is a brief summary of the output directed to **Print File**) (see Section 11.2):

- the optional arguments supplied via the option setting functions, if any;
- the basis file loaded, if any;
- a brief major iteration log (see Section);
- a brief minor iteration log (see Section);
- a summary of the final iterate.