

NAG Toolbox

nag_glopt_nlp_multistart_sqplsq (e05us)

1 Purpose

nag_glopt_nlp_multistart_sqplsq (e05us) is designed to find the global minimum of an arbitrary smooth sum of squares function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) by generating a number of different starting points and performing a local search from each using sequential quadratic programming.

2 Syntax

```
[x, objf, f, fjac, iter, c, cjac, clamda, istate, iopts, opts, user, info,
ifail] = nag_glopt_nlp_multistart_sqplsq(n, ncnln, a, bl, bu, y, confun,
objfun, npts, start, repeatl, nb, iopts, opts, 'm', m, 'nclin', nclin, 'user',
user)

[x, objf, f, fjac, iter, c, cjac, clamda, istate, iopts, opts, user, info,
ifail] = e05us(n, ncnln, a, bl, bu, y, confun, objfun, npts, start, repeatl, nb,
iopts, opts, 'm', m, 'nclin', nclin, 'user', user)
```

Before calling nag_glopt_nlp_multistart_sqplsq (e05us), the optional parameter arrays **iopts** and **opts** **must** be initialized for use with nag_glopt_nlp_multistart_sqplsq (e05us) by calling nag_glopt_optset (e05zk) with **optstr** set to ‘’. Optional parameters may subsequently be specified by calling nag_glopt_optset (e05zk) before the call to nag_glopt_nlp_multistart_sqplsq (e05us).

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 24: **user** was added as parameter to **start**; **r** is no longer returned.

3 Description

The local minimization method is nag_opt_lsqrcon_deriv (e04us). The problem is assumed to be stated in the following form:

$$\underset{x \in R^n}{\text{minimize}} F(x) = \frac{1}{2} \sum_{i=1}^m (y_i - f_i(x))^2 \quad \text{subject to} \quad l \leq \begin{pmatrix} x \\ A_L x \\ c(x) \end{pmatrix} \leq u, \quad (1)$$

where $F(x)$ (the *objective function*) is a nonlinear function which can be represented as the sum of squares of m subfunctions $(y_1 - f_1(x)), (y_2 - f_2(x)), \dots, (y_m - f_m(x))$, the y_i are constant, A_L is an n_L by n constant linear constraint matrix, and $c(x)$ is an n_N element vector of nonlinear constraint functions. (The matrix A_L and the vector $c(x)$ may be empty.) The objective function and the constraint functions are assumed to be smooth, i.e., at least twice-continuously differentiable. (This function will usually solve (1) if any isolated discontinuities are away from the solution.)

nag_glopt_nlp_multistart_sqplsq (e05us) solves a user-specified number of local optimization problems with different starting points. You may specify the starting points via the function **start**. If a random number generator is used to generate the starting points then the argument **repeatl** allows you to specify whether a repeatable set of points are generated or whether different starting points are generated on different calls. The resulting local minima are ordered and the best **nb** results returned in order of ascending values of the resulting objective function values at the minima. Thus the value returned in position 1 will be the best result obtained. If a sufficiently high number of different points are chosen then this is likely to be the global minimum.

4 References

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

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

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

n , the number of variables.

Constraint: **n** > 0.

2: **ncnln** – INTEGER

n_N , the number of nonlinear constraints.

Constraint: **ncnln** ≥ 0.

3: **a**(*lda*, :) – REAL (KIND=nag_wp) array

The first dimension of the array **a** must be at least **ncnln**.

The second dimension of the array **a** must be at least **n** if **ncnln** > 0, and at least 1 otherwise.

The matrix A_L of general linear constraints in (1). That is, the i th row contains the coefficients of the i th general linear constraint, for $i = 1, 2, \dots, \mathbf{ncnln}$.

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

4: **bl**(**n** + **ncnln** + **ncnln**) – REAL (KIND=nag_wp) array

5: **bu**(**n** + **ncnln** + **ncnln**) – REAL (KIND=nag_wp) array

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

Constraints:

$$\begin{aligned} \mathbf{bl}(j) &\leq \mathbf{bu}(j), \text{ for } j = 1, 2, \dots, \mathbf{n} + \mathbf{ncnln} + \mathbf{ncnln}; \\ \text{if } \mathbf{bl}(j) &= \mathbf{bu}(j) = \beta, |\beta| < \text{bigbnd}. \end{aligned}$$

6: **y**(**m**) – REAL (KIND=nag_wp) array

The coefficients of the constant vector y of the objective function.

7: **confun** – SUBROUTINE, supplied by the NAG Library or the user.

confun must calculate the vector $c(x)$ of nonlinear constraint functions and (optionally) its Jacobian ($= \frac{\partial c}{\partial x}$) for a specified n -element vector x . If there are no nonlinear constraints (i.e., **ncnln** = 0), **confun** will never be called by nag_glopt_nlp_multistart_sqp_lsq (e05us) and **confun** may be the string nag_opt_nlp1_dummy_confun (e04udm). (nag_opt_nlp1_dummy_confun (e04udm) is included in the NAG Toolbox.) If there are nonlinear constraints, the first call to **confun** will occur before the first call to **objfun**.

```
[mode, c, cjsl, user] = confun(mode, ncnln, n, ldcjsl, needc, x, cjsl,
nstate, user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **confun**. Only the following values need be assigned, for each value of i such that **needc**(i) > 0:

mode = 0
c(i), the i th nonlinear constraint.

mode = 1
All available elements in the i th row of **cjsl**.

mode = 2
c(i) and all available elements in the i th row of **cjsl**.

2: **ncnln** – INTEGER

n_N , the number of nonlinear constraints.

3: **n** – INTEGER

n , the number of variables.

4: **ldcjsl** – INTEGER

ldcjsl is the first dimension of the array **cjsl**.

5: **needc(ncnln)** – INTEGER array

The indices of the elements of **c** and/or **cjsl** that must be evaluated by **confun**. If **needc**(i) > 0, **c**(i) and/or the available elements of the i th row of **cjsl** (see argument **mode**) must be evaluated at x .

6: **x(n)** – REAL (KIND=nag_wp) array

x , the vector of variables at which the constraint functions and/or the available elements of the constraint Jacobian are to be evaluated.

7: **cjsl(ldcjsl, :)** – REAL (KIND=nag_wp) array

The second dimension of the array must be at least **n**.

cjsl may be regarded as a two-dimensional ‘slice’ of the three-dimensional array **cjac** of nag_glopt_nlp_multistart_sqplsq (e05us).

Unless **Derivative Level** = 2 or 3, the elements of **cjsl** are set to special values which enable nag_glopt_nlp_multistart_sqplsq (e05us) to detect whether they are changed by **confun**.

8: **nstate** – INTEGER

If **nstate** = 1 then nag_glopt_nlp_multistart_sqplsq (e05us) is calling **confun** for the first time on the current local optimization problem. This argument setting allows you to save computation time if certain data must be read or calculated only once.

9: **user** – INTEGER array

confun is called from nag_glopt_nlp_multistart_sqplsq (e05us) with the object supplied to nag_glopt_nlp_multistart_sqplsq (e05us).

Output Parameters

1: **mode** – INTEGER

May be set to a negative value if you wish to abandon the solution to the current local minimization problem. In this case `nag_glopt_nlp_multistart_sqplsq` (e05us) will move to the next local minimization problem.

2: **c(ncnln)** – REAL (KIND=nag_wp) array

If **needc**(*i*) > 0 and **mode** = 0 or 2, **c**(*i*) must contain the value of $c_i(x)$. The remaining elements of **c**, corresponding to the non-positive elements of **needc**, need not be set.

3: **cjst(ldecjst,:)** – REAL (KIND=nag_wp) array

The second dimension of the array will be **n**.

cjst may be regarded as a two-dimensional ‘slice’ of the three-dimensional array **cjac** of `nag_glopt_nlp_multistart_sqplsq` (e05us).

If **needc**(*i*) > 0 and **mode** = 1 or 2, the *i*th row of **cjst** must contain the available elements of the vector ∇c_i given by

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

where $\frac{\partial c_i}{\partial x_j}$ is the partial derivative of the *i*th constraint with respect to the *j*th variable,

evaluated at the point *x*. See also the argument **nstate**. The remaining rows of **cjst**, corresponding to non-positive elements of **needc**, need not be set.

If all elements of the constraint Jacobian are known (i.e., **Derivative Level** = 2 or 3; note the default is **Derivative Level** = 3), any constant elements may be assigned to **cjst** one time only at the start of each local optimization. An element of **cjst** that is not subsequently assigned in **confun** will retain its initial value throughout the local optimization. Constant elements may be loaded into **cjst** during the first call to **confun** for the local optimization (signalled by the value **nstate** = 1). The ability to preload constants is useful when many Jacobian elements are identically zero, in which case **cjst** may be initialized to zero and nonzero elements may be reset by **confun**.

Note that constant nonzero elements do affect the values of the constraints. Thus, if **cjst**(*i*, *j*) is set to a constant value, it need not be reset in subsequent calls to **confun**, but the value **cjst**(*i*, *j*) × **x**(*j*) must nonetheless be added to **c**(*i*). For example, if **cjst**(1, 1) = 2 and **cjst**(1, 2) = −5 then the term 2 × **x**(1) − 5 × **x**(2) must be included in the definition of **c**(1).

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

4: **user** – INTEGER array

confun should be tested separately before being used in conjunction with `nag_glopt_nlp_multistart_sqplsq` (e05us). See also the description of the optional parameter **Verify**.

8: **objfun** – SUBROUTINE, supplied by the user.

objfun must calculate either the *i*th element of the vector $f(x) = (f_1(x), f_2(x), \dots, f_m(x))^T$ or all *m* elements of $f(x)$ and (optionally) its Jacobian $\left(= \frac{\partial f}{\partial x} \right)$ for a specified *n*-element vector *x*.

```
[mode, f, fjsl, user] = objfun(mode, m, n, ldfjsl, needfi, x, fjsl,
nstate, user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode = 0 and **needfi** = i , where $i > 0$
 $f(i)$.

mode = 0 and **needfi** < 0
 f .

mode = 1 and **needfi** < 0
All available elements of **fjsl**.

mode = 2 and **needfi** < 0
 f and all available elements of **fjsl**.

2: **m** – INTEGER

m , the number of subfunctions.

3: **n** – INTEGER

n , the number of variables.

4: **ldfjsl** – INTEGER

ldfjsl is the first dimension of the array **fjsl**.

5: **needfi** – INTEGER

If **needfi** = $i > 0$, only the i th element of $f(x)$ needs to be evaluated at x ; the remaining elements need not be set. This can result in significant computational savings when $m \gg n$.

6: **x(n)** – REAL (KIND=nag_wp) array

x , the vector of variables at which the objective function and/or all available elements of its gradient are to be evaluated.

7: **fjsl(ldfjsl,:)** – REAL (KIND=nag_wp) array

The first dimension of the array **fjsl** must be at least .

The second dimension of the array **fjsl** must be at least **n**.

fjsl may be regarded as a two-dimensional ‘slice’ of the three-dimensional array **fjac** of nag_glopt_nlp_multistart_sqp_lsqr (e05us).

Is set to a special value.

8: **nstate** – INTEGER

If **nstate** = 1 then nag_glopt_nlp_multistart_sqp_lsqr (e05us) is calling **objfun** for the first time on the current local optimization problem. This argument setting allows you to save computation time if certain data must be read or calculated only once.

9: **user** – INTEGER array

objfun is called from `nag_glopt_nlp_multistart_sqp_lsq` (e05us) with the object supplied to `nag_glopt_nlp_multistart_sqp_lsq` (e05us).

Output Parameters

1: **mode** – INTEGER

May be set to a negative value if you wish to abandon the solution to the current local minimization problem. In this case `nag_glopt_nlp_multistart_sqp_lsq` (e05us) will move to the next local minimization problem.

2: **f(m)** – REAL (KIND=nag_wp) array

If **mode** = 0 and **needfi** = $i > 0$, **f**(i) must contain the value of f_i at x .

If **mode** = 0 or 2 and **needfi** < 0, **f**(i) must contain the value of f_i at x , for $i = 1, 2, \dots, m$.

3: **fjst**(**ldfjst**, :) – REAL (KIND=nag_wp) array

The first dimension of the array **fjst** will be .

The second dimension of the array **fjst** will be **n**.

fjst may be regarded as a two-dimensional ‘slice’ of the three-dimensional array **fjac** of `nag_glopt_nlp_multistart_sqp_lsq` (e05us).

If **mode** = 1 or 2 and **needfi** < 0, the i th row of **fjst** must contain the available elements of the vector ∇f_i given by

$$\nabla f_i = (\partial f_i / \partial x_1, \partial f_i / \partial x_2, \dots, \partial f_i / \partial x_n)^T,$$

evaluated at the point x . See also the argument **nstate**.

4: **user** – INTEGER array

objfun should be tested separately before being used in conjunction with `nag_glopt_nlp_multistart_sqp_lsq` (e05us). See also the description of the optional parameter **Verify**.

9: **npts** – INTEGER

The number of different starting points to be generated and used. The more points used, the more likely that the best returned solution will be a global minimum.

Constraint: $1 \leq \mathbf{nb} \leq \mathbf{npts}$.

10: **start** – SUBROUTINE, supplied by the NAG Library or the user.

start must calculate the **npts** starting points to be used by the local optimizer. If you do not wish to write a function specific to your problem then `nag_glopt_multistart_start_points` (e05ucz) may be used as the actual argument. `nag_glopt_multistart_start_points` (e05ucz) is supplied in the NAG Toolbox and uses the NAG quasi-random number generators to distribute starting points uniformly across the domain. It is affected by the value of **repeat1**.

```
[quas, user, mode] = start(npts, quas, n, repeat1, bl, bu, user, mode)
```

Input Parameters

1: **npts** – INTEGER

Indicates the number of starting points.

- 2: **quas**(**n**, **npts**) – REAL (KIND=nag_wp) array
All elements of **quas** will have been set to zero, so only nonzero values need be set subsequently.
- 3: **n** – INTEGER
The number of variables.
- 4: **repeat1** – LOGICAL
Specifies whether a repeatable or non-repeatable sequence of points are to be generated.
- 5: **bl**(**n**) – REAL (KIND=nag_wp) array
The lower bounds on the variables. These may be used to ensure that the starting points generated in some sense ‘cover’ the region, but there is no requirement that a starting point be feasible.
- 6: **bu**(**n**) – REAL (KIND=nag_wp) array
The upper bounds on the variables. (See **bl**.)
- 7: **user** – INTEGER array
start is called from nag_glopt_nlp_multistart_sqp_ls (e05us) with the object supplied to nag_glopt_nlp_multistart_sqp_ls (e05us).
- 8: **mode** – INTEGER
mode will contain 0.

Output Parameters

- 1: **quas**(**n**, **npts**) – REAL (KIND=nag_wp) array
Must contain the starting points for the **npts** local minimizations, i.e., **quas**(*j*, *i*) must contain the *j*th component of the *i*th starting point.
- 2: **user** – INTEGER array
- 3: **mode** – INTEGER
If you set **mode** to a negative value then nag_glopt_nlp_multistart_sqp_ls (e05us) will terminate immediately with **ifail** = 9.
- 11: **repeat1** – LOGICAL
Is passed as an argument to **start** and may be used to initialize a random number generator to a repeatable, or non-repeatable, sequence. See Section 9 for more detail.
- 12: **nb** – INTEGER
The number of solutions to be returned. The function saves up to **nb** local minima ordered by increasing value of the final objective function. If the defining criterion for ‘best solution’ is only that the value of the objective function is as small as possible then **nb** should be set to 1. However, if you want to look at other solutions that may have desirable properties then setting **nb** > 1 will produce **nb** local minima, ordered by increasing value of their objective functions at the minima.
Constraint: $1 \leq \mathbf{nb} \leq \mathbf{npts}$.

- 13: **iopts**(740) – INTEGER array
 14: **opts**(485) – REAL (KIND=nag_wp) array

The arrays **iopts** and **opts** **must not** be altered between calls to any of the functions nag_glopt_nlp_multistart_sqp_lsqr (e05us) and nag_glopt_optset (e05zk).

5.2 Optional Input Parameters

- 1: **m** – INTEGER
Default: the dimension of the array **y**.
 m , the number of subfunctions associated with $F(x)$.
Constraint: **m** > 0.
- 2: **nclin** – INTEGER
Default: the first dimension of the array **a**.
 n_L , the number of general linear constraints.
Constraint: **nclin** ≥ 0.
- 3: **user** – INTEGER array
user is not used by nag_glopt_nlp_multistart_sqp_lsqr (e05us), but is passed to **confun**, **objfun** and **start**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

- 1: **x**(ldx,:) – REAL (KIND=nag_wp) array
 The first dimension of the array **x** will be **n**.
 The second dimension of the array **x** will be **nb**.
x(j, i) contains the final estimate of the i th solution, for $j = 1, 2, \dots, \mathbf{n}$.
- 2: **objf**(nb) – REAL (KIND=nag_wp) array
objf(i) contains the value of the objective function at the final iterate for the i th solution.
- 3: **f**(m,:) – REAL (KIND=nag_wp) array
 The second dimension of the array **f** will be **nb**.
f(j, i) contains the value of the j th function f_j at the final iterate, for $j = 1, 2, \dots, \mathbf{m}$, for the i th solution, for $i = 1, 2, \dots, \mathbf{nb}$.
- 4: **fjac**(ldfjac, sdfjac,:) – REAL (KIND=nag_wp) array
 The last dimension of the array **fjac** will be **nb**
 $sdfjac = \mathbf{n}$.
 For the i th returned solution, the Jacobian matrix of the functions f_1, f_2, \dots, f_m at the final iterate, i.e., **fjac**(k, j, i) contains the partial derivative of the k th function with respect to the j th variable, for $k = 1, 2, \dots, \mathbf{m}$, $j = 1, 2, \dots, \mathbf{n}$ and $i = 1, 2, \dots, \mathbf{nb}$. (See also the discussion of argument **fjst** under **objfun**.)
- 5: **iter**(nb) – INTEGER array
iter(i) contains the number of major iterations performed to obtain the i th solution. If less than **nb** solutions are returned then **iter**(nb) contains the number of starting points that have resulted

in a converged solution. If this is close to **npts** then this might be indicative that fewer than **nb** local minima exist.

- 6: **c**(*ldc*,:) – REAL (KIND=nag_wp) array

The first dimension of the array **c** will be **ncnln**.

The second dimension of the array **c** will be **nb**.

If **ncnln** > 0, **c**(*j*, *i*) contains the value of the *j*th nonlinear constraint function c_j at the final iterate, for the *i*th solution, for $j = 1, 2, \dots, \text{ncnln}$.

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

- 7: **cjac**(*ldcjac*, *sdcjac*,:) – REAL (KIND=nag_wp) array

The last dimension of the array **cjac** will be **nb**

If **ncnln** > 0, **cjac** contains the Jacobian matrices of the nonlinear constraint functions at the final iterate for each of the returned solutions, i.e., **cjac**(*k*, *j*, *i*) contains the partial derivative of the *k*th constraint function with respect to the *j*th variable, for $k = 1, 2, \dots, \text{ncnln}$ and $j = 1, 2, \dots, \text{n}$, for the *i*th solution. (See the discussion of argument **cjsl** under **confun**.)

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

- 8: **clamda**(*ldclda*,:) – REAL (KIND=nag_wp) array

The first dimension of the array **clamda** will be **n** + **nclin** + **ncnln**.

The second dimension of the array **clamda** will be **nb**.

The values of the QP multipliers from the last QP subproblem solved for the *i*th solution. **clamda**(*j*, *i*) should be non-negative if **istate**(*j*, *i*) = 1 and non-positive if **istate**(*j*, *i*) = 2.

- 9: **istate**(*listat*,:) – INTEGER array

The first dimension of the array **istate** will be **n** + **nclin** + **ncnln**.

The second dimension of the array **istate** will be **nb**.

istate(*j*, *i*) contains the status of the constraints in the QP working set for the *i*th solution. The significance of each possible value of **istate**(*j*, *i*) is as follows:

istate (<i>j</i> , <i>i</i>)	Meaning
0	The constraint is satisfied to within the feasibility tolerance, but is not in the QP working set.
1	This inequality constraint is included in the QP working set at its lower bound.
2	This inequality constraint is included in the QP working set at its upper bound.
3	This constraint is included in the QP working set as an equality. This value of istate can occur only when bl (<i>j</i>) = bu (<i>j</i>).

- 10: **iopts**(740) – INTEGER array

- 11: **opts**(485) – REAL (KIND=nag_wp) array

- 12: **user** – INTEGER array

- 13: **info**(**nb**) – INTEGER array

If **ifail** = 0, **info**(*i*) does not contain an error value returned by nag_opt_lsq_gencon_deriv (e04us) .

If **ifail** = 8 on exit, then not all **nb** solutions have been found, and **info**(**nb**) contains the number of solutions actually found.

14: **ifail** – INTEGER

ifail = 0 unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: `nag_glopt_nlp_multistart_sqp_lsq` (e05us) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail = 1

An input value is incorrect. One or more of the following requirements are violated:

Constraint: $1 \leq \mathbf{nb} \leq \mathbf{npts}$.

Constraint: $\mathbf{bl}(i) \leq \mathbf{bu}(i)$, for all i .

Constraint: if $\mathbf{ncnln} > 0$, $\mathbf{sdcjac} \geq \mathbf{n}$.

Constraint: $\mathbf{lda} \geq \mathbf{nclin}$.

Constraint: $\mathbf{ldcjac} \geq \mathbf{ncnln}$.

Constraint: $\mathbf{ldclda} \geq \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$.

Constraint: $\mathbf{ldc} \geq \mathbf{ncnln}$.

Constraint: $\mathbf{ldfjac} \geq \mathbf{m}$.

Constraint: $\mathbf{ldx} \geq \mathbf{n}$.

Constraint: $\mathbf{listat} \geq \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$.

Constraint: $\mathbf{m} > 0$.

Constraint: $\mathbf{n} > 0$.

Constraint: $\mathbf{nclin} \geq 0$.

Constraint: $\mathbf{ncnln} \geq 0$.

Constraint: $\mathbf{sdfjac} \geq \mathbf{n}$.

ifail = 2

*`nag_glopt_nlp_multistart_sqp_lsq` (e05us) has terminated without finding any solutions. The majority of calls to the local optimizer have failed to find a feasible point for the linear constraints and bounds, which means that either no feasible point exists for the given value of the optional parameter **Linear Feasibility Tolerance** (default value $\sqrt{\text{macheps}}$, where *macheps* is the **machine precision**), or no feasible point could be found in the number of iterations specified by the optional parameter **Minor Iteration Limit**. You should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ , you should ensure that the value of the optional parameter **Linear Feasibility Tolerance** is greater than σ . For example, if all elements of A_L are of order unity and are accurate to only three decimal places, **Linear Feasibility Tolerance** should be at least 10^{-3} .*

No solution obtained. Linear constraints may be infeasible.

ifail = 3

`nag_glopt_nlp_multistart_sqp_lsq` (e05us) has failed to find any solutions. The majority of local optimizations could not find a feasible point for the nonlinear constraints. The problem may have no feasible solution. This behaviour will occur if there is no feasible point for the nonlinear constraints. (However, there is no general test that can determine whether a feasible point exists for a set of nonlinear constraints.)

No solution obtained. Nonlinear constraints may be infeasible.

ifail = 4

nag_glopt_nlp_multistart_sqp_lsq (e05us) has failed to find any solutions. The majority of local optimizations have failed because the limiting number of iterations have been reached.

No solution obtained. Many potential solutions reach iteration limit.

ifail = 7

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

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

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

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

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

ifail = 8 (*warning*)

Only *<value>* solutions obtained.

Not all **nb** solutions have been found. **info(nb)** contains the number actually found.

ifail = 9

User terminated computation from **start** procedure. **mode** = *<value>*.

*If nag_glopt_multistart_start_points (e05ucz) has been used as an actual argument for **start** then the message displayed, when **ifail** = 0 or -1 on entry to nag_glopt_nlp_multistart_sqp_lsq (e05us), will have the following meaning:*

998 failure to allocate space, a smaller value of NPTS should be tried.

997 an internal error has occurred. Please contact NAG for assistance.

ifail = 10

Failed to initialize optional parameter arrays.

ifail = -99

An unexpected error has been triggered by this routine. Please contact NAG.

ifail = -399

Your licence key may have expired or may not have been installed correctly.

ifail = -999

Dynamic memory allocation failed.

7 Accuracy

If **ifail** = 0 on exit and the value of **info**(*i*) = 0, then the vector returned in the array **x** for solution *i* is an estimate of the solution to an accuracy of approximately **Optimality Tolerance**.

8 Further Comments

You should be wary of requesting much intermediate output from the local optimizer, since large volumes may be produced if **npts** is large.

The auxiliary routine `nag_glopt_multistart_start_points` (e05ucz) makes use of the NAG quasi-random Sobol generator (`nag_rand_quasi_init` (g05yl) and `nag_rand_quasi_uniform` (g05ym)). If `nag_glopt_multistart_start_points` (e05ucz) is used as the actual argument for **start** and **repeat1** = *false* then a randomly chosen value for **iskip** is used, otherwise **iskip** is set to 100. If **repeat1** is set to *false* and the program is executed several times, each time producing the same best answer, then there is increased probability that this answer is a global minimum. However, if it is important that identical results be obtained on successive runs, then **repeat1** should be set to *true*.

8.1 Description of the Printed Output

See Section 9.1 in `nag_opt_lsq_gencon_deriv` (e04us).

9 Example

This example is based on Problem 57 in Hock and Schittkowski (1981) and involves the minimization of the sum of squares function

$$F(x) = \frac{1}{2} \sum_{i=1}^{44} (y_i - f_i(x))^2,$$

where

$$f_i(x) = x_1 + (0.49 - x_1)e^{-x_2(a_i-8)}$$

and

<i>i</i>	<i>y_i</i>	<i>a_i</i>	<i>i</i>	<i>y_i</i>	<i>a_i</i>
1	0.49	8	23	0.41	22
2	0.49	8	24	0.40	22
3	0.48	10	25	0.42	24
4	0.47	10	26	0.40	24
5	0.48	10	27	0.40	24
6	0.47	10	28	0.41	26
7	0.46	12	29	0.40	26
8	0.46	12	30	0.41	26
9	0.45	12	31	0.41	28
10	0.43	12	32	0.40	28
11	0.45	14	33	0.40	30
12	0.43	14	34	0.40	30
13	0.43	14	35	0.38	30
14	0.44	16	36	0.41	32
15	0.43	16	37	0.40	32
16	0.43	16	38	0.40	34
17	0.46	18	39	0.41	36
18	0.45	18	40	0.38	36
19	0.42	20	41	0.40	38
20	0.42	20	42	0.40	38
21	0.43	20	43	0.39	40
22	0.41	22	44	0.39	42

subject to the bounds

$$\begin{aligned} x_1 &\geq 0.4 \\ x_2 &\geq -4.0 \end{aligned}$$

to the general linear constraint

$$x_1 + x_2 \geq 1.0$$

and to the nonlinear constraint

$$0.49x_2 - x_1x_2 \geq 0.09.$$

The optimal solution (to five figures) is

$$x^* = (0.41995, 1.28484)^T,$$

and $F(x^*) = 0.01423$. The nonlinear constraint is active at the solution.

9.1 Program Text

```
function e05us_example

fprintf('e05us example results\n\n');

npts = nag_int(1);
repeat = true;
nclin = nag_int(1);
ncnln = nag_int(1);
nb = nag_int(1);
n = nag_int(2);

bl = [0.4; -4.0; 1.0; 0.0];
bu = [1e25; 1e25; 1e25; 1e25];

a = [1, 1];
x = [0.4; 0];
y = [0.49; 0.49; 0.48; 0.47; 0.48; 0.47; 0.46; 0.46; 0.45; 0.43; 0.45; ...
     0.43; 0.43; 0.44; 0.43; 0.43; 0.46; 0.45; 0.42; 0.42; 0.43; 0.41; ...
     0.41; 0.40; 0.42; 0.40; 0.40; 0.41; 0.40; 0.41; 0.41; 0.40; 0.40; ...
     0.40; 0.38; 0.41; 0.40; 0.40; 0.41; 0.38; 0.40; 0.40; 0.39; 0.39];

% Initialise e05us
iopts = zeros(740, 1, nag_int_name);
opts = zeros(485, 1);
[iopts, opts, ifail] = e05zk(...
    'Initialize = e05us', iopts, opts);
[iopts, opts, ifail] = e05zk(...
    'List', iopts, opts);
[iopts, opts, ifail] = e05zk(...
    'print level = 10', iopts, opts);

wstat = warning();
warning('OFF');
% Solve the problem
[x, objf, f, fjac, iter, c, cjac, clamda, istate, ...
    iopts, opts, user, info, ifail] = ...
    e05us(...
        n, ncnln, a, bl, bu, y, @confun, @objfun, ...
        npts, @start, repeat, nb, iopts, opts);

warning(wstat);

if ifail == 8
    l = double(info(nb));
    fprintf('\nOnly %d solutions found\n', info(nb));
else
    l = double(nb);
end
l = min(l,3);

for i=1:l
    fprintf('\nSolution number %d\n\n', i);
    fprintf('e04us returned with ifail = %d\n\n', info(i));

    fprintf(' Variable Istate          Value Lagrange Multiplier\n');
    for j=1:n
        fprintf(' %3d%9d%19.6g%13.4g\n', j, istate(j,i), x(j,i), clamda(j,i));
    end
end
```

```

if nclin > 0
    ax = a*x(:,i);
    fprintf('\n L Con      Istate      Value Lagrange Multiplier\n');
    for k=n+1:n+nclin
        j=k-n;
        fprintf(' %3d%9d%19.6g%13.4g\n', j, istate(k,i), ax(j), clamda(k,i));
    end
end

if ncnln > 0
    fprintf('\n NL Con      Istate      Value Lagrange Multiplier\n');
    for k=n+nclin+1:n+nclin+ncnln
        j=k-n-nclin;
        fprintf(' %3d%9d%19.6g%13.4g\n', j, istate(k,i), c(j,i), clamda(j,i));
    end
end

fprintf('\nFinal objective value = %15.7g\n', objf(i));
fprintf('clamda: ');
disp(transpose(clamda(1:double(n+nclin+ncnln),i)));
fprintf('\n ----- \n');
end

function [mode, c, cjsl, user] = confun(mode, ncnln, n, ldcjsl, ...
                                       needc, x, cjsl, nstate, user)

    c = zeros(ncnln, 1);

    if (nstate == 1)
        % First call to confun. Set all Jacobian elements to zero.
        % Note that this will only work when 'Derivative Level = 3'
        % (the default).
        cjsl(1:double(ncnln),1:double(n)) = 0;
    end

    if ( needc(1) > 0 )

        if (mode==0 || mode==2)
            c(1) = -0.09 - x(1)*x(2) + 0.49*x(2);
        end

        if (mode==1 || mode==2)
            cjsl(1,1) = -x(2);
            cjsl(1,2) = -x(1) + 0.49;
        end
    end

end

function [mode, f, fjac, user] = objfun(mode, m, n, ldfj, needfi, x, fjac, ...
                                       nstate, user)

    % Evaluate the subfunctions and their 1st derivatives.

    a = [ 8; 8; 10; 10; 10; 10; 12; 12; 12; 12; 14;
          14; 14; 16; 16; 16; 18; 18; 20; 20; 20; 22;
          22; 22; 24; 24; 24; 26; 26; 26; 28; 28; 30;
          30; 30; 32; 32; 34; 36; 36; 38; 38; 40; 42];

    f = zeros(m, 1);

    if needfi > 0
        f(needfi) = x(1) + (0.49-x(1))*exp(-x(2)*(a(needfi)-8.0));
        return;
    end

    temp = exp(-x(2)*(a-8.0));

    if (mode==0 || mode==2)
        f = x(1) + (0.49-x(1))*temp;
    end

    if (mode==1 || mode==2)

```

```

    fjac(:,1) = 1.0 - temp;
    fjac(:,2) = -(0.49-x(1)).*(a-8.0).*temp;
end

function [quas, mode, user] = start(npts, quas, n, repeat, bl, bu, mode, user)
    quas(1, 1) = 0.4;
    quas(2, 1) = 0.0;

```

9.2 Program Results

e05us example results

Calls to E05ZKF

```

LIST
PRINT LEVEL = 10

```

*** E05USF

Parameters

Linear constraints.....	1	Variables.....	2
Nonlinear constraints..	1	Subfunctions.....	44
Infinite bound size....	1.00E+20	COLD start.....	
Infinite step size.....	1.00E+20	EPS (machine precision)	1.11E-16
Step limit.....	2.00E+00	Hessian.....	NO
Linear feasibility.....	1.05E-08	Crash tolerance.....	1.00E-02
Nonlinear feasibility..	1.05E-08	Optimality tolerance...	3.26E-12
Line search tolerance..	9.00E-01	Function precision.....	4.37E-15
Derivative level.....	3	Monitoring file.....	-1
Verify level.....	0		
Major iterations limit.	50	Major print level.....	10
Minor iterations limit.	50	Minor print level.....	0
J'J initial Hessian....		Reset frequency.....	2
Workspace provided is	IWORK(9),	WORK(306).	
To solve problem we need	IWORK(9),	WORK(306).	

Verification of the constraint gradients.

The constraint Jacobian seems to be ok.

The largest relative error was 1.89E-08 in constraint 1

Verification of the objective gradients.

The objective Jacobian seems to be ok.

The largest relative error was 1.04E-08 in subfunction 3

Maj	Mnr	Step	Merit	Function	Norm Gz	Violtn	Cond	Hx
0	2	0.0E+00	2.224070E-02	8.5E-02	3.6E-02	1.0E+00		
1	1	1.0E+00	1.455402E-02	1.5E-03	9.8E-03	1.0E+00		
2	1	1.0E+00	1.436491E-02	4.9E-03	7.2E-04	1.0E+00		
3	1	1.0E+00	1.427013E-02	2.9E-03	9.2E-06	1.0E+00		
4	1	1.0E+00	1.422989E-02	1.6E-04	3.6E-05	1.0E+00		
5	1	1.0E+00	1.422983E-02	5.4E-07	6.4E-08	1.0E+00		
6	1	1.0E+00	1.422983E-02	3.4E-09	9.8E-13	1.0E+00		

Exit from NP problem after 6 major iterations,
8 minor iterations.

Varbl	State		Value	Lower Bound	Upper Bound	Lagr Mult	Slack
V	1	FR	0.419953	0.400000	None	.	1.9953E-02
V	2	FR	1.28485	-4.00000	None	.	5.285

L Con	State		Value	Lower Bound	Upper Bound	Lagr Mult	Slack
L	1	FR	1.70480	1.00000	None	.	0.7048

N Con	State		Value	Lower Bound	Upper Bound	Lagr Mult	Slack
N	1	LL	-9.767742E-13	.	None	3.3358E-02	-9.7677E-13

Exit E05USF - Optimal solution found.

Final objective value = 0.1422983E-01

Solution number 1

e04us returned with ifail = 0

Variable	Istate	Value	Lagrange	Multiplier
1	0	0.419953		0
2	0	1.28485		0

L Con	Istate	Value	Lagrange	Multiplier
1	0	1.7048		0

NL Con	Istate	Value	Lagrange	Multiplier
1	1	-9.76774e-13		0

Final objective value = 0.01422983

clamda: 0 0 0 0.0334

10 Algorithmic Details

See Section 11 in nag_opt_lsq_gencon_deriv (e04us).

11 Optional Parameters

Several optional parameters in nag_glopt_nlp_multistart_sqp_lsq (e05us) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_glopt_nlp_multistart_sqp_lsq (e05us) these optional parameters have associated *default values* that are appropriate for most problems. Therefore you need only specify those optional parameters whose values are to be different from their default values.

Optional parameters may be specified by calling nag_glopt_optset (e05zk) before a call to nag_glopt_nlp_multistart_sqp_lsq (e05us). Before calling nag_glopt_nlp_multistart_sqp_lsq (e05us), the optional parameter arrays **iopts** and **opts** **must** be initialized for use with nag_glopt_nlp_multistart_sqp_lsq (e05us) by calling nag_glopt_optset (e05zk) with **optstr** set to ‘’.

All optional parameters not specified are set to their default values. Optional parameters specified are unaltered by nag_glopt_nlp_multistart_sqp_lsq (e05us) (unless they define invalid values) and so remain in effect for subsequent calls to nag_glopt_nlp_multistart_sqp_lsq (e05us).

See Section 12 in nag_opt_lsq_gencon_deriv (e04us) for full details.

The **Warm Start** option of nag_opt_lsq_gencon_deriv (e04us) is not a valid option for use with nag_glopt_nlp_multistart_sqp_lsq (e05us).

nag_glopt_nlp_multistart_sqp_lsq (e05us) supports two options that are distinct from those of nag_opt_lsq_gencon_deriv (e04us):

Punch Unit*i*

Default = 6

This option allows you to send information arising from an appropriate setting of **Out_Level** to be sent to the Fortran unit number defined by **Punch Unit**. If you wish this file to be different to the standard output unit (6) where other output is displayed then this file should be attached by calling `nag_file_open` (x04ac) prior to calling `nag_glopt_nlp_multistart_sqp_lsq` (e05us).

Out_Level*i*

Default = 0

This option defines the amount of extra information to be sent to the Fortran unit number defined by **Punch Unit**. The possible choices for *i* are the following:

<i>i</i>	Meaning
0	No extra output.
1	Updated solutions only. This is useful during long runs to observe progress.
2	Successful start points only. This is useful to save the starting points that gave rise to the final solution.
3	Both updated solutions and successful start points.

12 Description of Monitoring Information

See Section 13 in `nag_opt_lsq_gencon_deriv` (e04us).
