NAG Toolbox

nag_glopt_nlp_multistart_sqp (e05uc)

1 Purpose

nag_glopt_nlp_multistart_sqp (e05uc) is designed to find the global minimum of an arbitrary smooth 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, objgrd, iter, c, cjac, r, clamda, istate, iopts, opts, user, info,
ifail] = nag_glopt_nlp_multistart_sqp(n, ncnln, a, bl, bu, confun, objfun, npts,
start, repeat, nb, iopts, opts, 'nclin', nclin, 'user', user)
[x, objf, objgrd, iter, c, cjac, r, clamda, istate, iopts, opts, user, info,
ifail] = e05uc(n, ncnln, a, bl, bu, confun, objfun, npts, start, repeat, nb,
iopts, opts, 'nclin', nclin, 'user', user)
```

Before calling nag_glopt_nlp_multistart_sqp (e05uc), the optional parameter arrays **iopts** and **opts must** be initialized for use with nag_glopt_nlp_multistart_sqp (e05uc) by calling nag_glopt_optset (e05zk) with **optstr** set to ‘’. Optional parameters may be specified by calling nag_glopt_optset (e05zk) before the call to nag_glopt_nlp_multistart_sqp (e05uc).

3 Description

The problem is assumed to be stated in the following form:

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

where $F(x)$ (the *objective function*) is a nonlinear function, A_L is an n_L by n 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 there are only isolated discontinuities away from the solution.)

nag_glopt_nlp_multistart_sqp (e05uc) 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 **repeat** 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 sufficient number of different points are chosen then this is likely to be be the global minimum. Please note that the default version of **start** uses a random number generator to generate the starting points.

4 References

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

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

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

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

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

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

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

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

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

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

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

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

n, the number of variables.

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

2: **ncnln** – INTEGER

n_N, the number of nonlinear constraints.

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

3: $\mathbf{a}(lda, :)$ – REAL (KIND=nag_wp) array

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

The second dimension of the array **a** must be at least **n** if $\mathbf{nclin} > 0$, and at least 1 otherwise.

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

If $\mathbf{nclin} = 0$, the array **a** is not referenced.

4: $\mathbf{bl}(\mathbf{n} + \mathbf{nclin} + \mathbf{ncnln})$ – REAL (KIND=nag_wp) array

5: $\mathbf{bu}(\mathbf{n} + \mathbf{nclin} + \mathbf{ncnln})$ – 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 $\mathbf{bl}(j) \leq -bigbnd$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}(j) \geq bigbnd$; the default value of $bigbnd$ is 10^{20}, but this may be changed by the optional parameter **Infinite Bound Size**. To specify the jth constraint as an equality, set $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, say, where $|\beta| < bigbnd$.

Constraints:

$\mathbf{bl}(j) \leq \mathbf{bu}(j)$, for $j = 1, 2, \ldots, \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$;
if $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, $|\beta| < bigbnd$.

6: **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 ($= \dfrac{\partial c}{\partial x}$) for a specified n-element vector x. If there are no nonlinear constraints (i.e., $\mathbf{ncnln} = 0$), **confun** will never be called by nag_glopt_nlp_multistart_sqp (e05uc) 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 $\mathbf{needc}(i) > 0$:

mode $= 0$
$\mathbf{c}(i)$.

mode $= 1$
All available elements in the ith row of **cjsl**.

mode $= 2$
$\mathbf{c}(i)$ and all available elements in the ith 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 same value as $ldcjac$ in the call to nag_glopt_nlp_multistart_sqp (e05uc).

5: **needc**(**ncnln**) – INTEGER array

The indices of the elements of **c** and/or **cjsl** that must be evaluated by **confun**. If $\mathbf{needc}(i) > 0$, $\mathbf{c}(i)$ and/or the available elements of the ith 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**, **n**) – REAL (KIND=nag_wp) array

cjsl may be regarded as a two-dimensional 'slice' of the three-dimensional array **cjac** of nag_glopt_nlp_multistart_sqp (e05uc).

Unless **Derivative Level** $= 2$ or 3 (the default setting is **Derivative Level** $= 3$, the elements of **cjsl** are set to special values which enable nag_glopt_nlp_multistart_sqp (e05uc) to detect whether they are changed by **confun**.

8: **nstate** – INTEGER

If **nstate** $= 1$ then nag_glopt_nlp_multistart_sqp (e05uc) 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 calculated only once.

9: **user** – INTEGER array

confun is called from nag_glopt_nlp_multistart_sqp (e05uc) with the object supplied to nag_glopt_nlp_multistart_sqp (e05uc).

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 (e05uc) will move to the next local minimization problem.

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

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

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

cjsl may be regarded as a two-dimensional 'slice' of the three-dimensional array **cjac** of nag_glopt_nlp_multistart_sqp (e05uc).

If **needc**$(k) > 0$ and **mode** $= 1$ or 2, the kth row of **cjsl** must contain the available elements of the vector ∇c_k given by

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

where $\dfrac{\partial c_k}{\partial x_j}$ is the partial derivative of the kth constraint with respect to the jth variable, evaluated at the point x. See also the argument **nstate**. The remaining rows of **cjsl**, 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), any constant elements may be assigned to **cjsl** one time only at the start of each local optimization. An element of **cjsl** that is not subsequently assigned in **confun** will retain its initial value throughout the local optimization. Constant elements may be loaded into **cjsl** 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 **cjsl** 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 **cjsl**(k, j) is set to a constant value, it need not be reset in subsequent calls to **confun**, but the value **cjsl**$(k, j) \times$ **x**(j) must nonetheless be added to **c**(k). For example, if **cjsl**$(1, 1) = 2$ and **cjsl**$(1, 2) = -5$ then the term $2 \times$ **x**$(1) - 5 \times$ **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 **cjsl** 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 **cjsl**, 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_sqp (e05uc). See also the description of the optional parameter **Verify**.

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

objfun must calculate the objective function $F(x)$ and (optionally) its gradient $g(x) = \dfrac{\partial F}{\partial x}$ for a specified n-vector x.

```
[mode, objf, objgrd, user] = objfun(mode, n, x, objgrd, 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 $= 0$
objf.

mode $= 1$
All available elements of **objgrd**.

mode $= 2$
objf and all available elements of **objgrd**.

2: **n** – INTEGER

n, the number of variables.

3: **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.

4: **objgrd**(**n**) – REAL (KIND=nag_wp) array

The elements of **objgrd** are set to special values which enable nag_glopt_nlp_multistart_sqp (e05uc) to detect whether they are changed by **objfun**.

5: **nstate** – INTEGER

If **nstate** $= 1$ then nag_glopt_nlp_multistart_sqp (e05uc) 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 calculated only once.

6: **user** – INTEGER array

objfun is called from nag_glopt_nlp_multistart_sqp (e05uc) with the object supplied to nag_glopt_nlp_multistart_sqp (e05uc).

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 (e05uc) will move to the next local minimization problem.

2: **objf** – REAL (KIND=nag_wp)

If **mode** $= 0$ or 2, **objf** must be set to the value of the objective function at x.

3: **objgrd**(**n**) – REAL (KIND=nag_wp) array

If **mode** $= 1$ or 2, **objgrd** must return the available elements of the gradient evaluated at x.

4: **user** – INTEGER array

objfun should be tested separately before being used in conjunction with nag_glopt_nlp_multistart_sqp (e05uc). See also the description of the optional parameter **Verify**.

8: **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: $1 \leq$ **nb** $\leq$ **npts**.

9: **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 **repeat**.

```
[quas, user, mode] = start(npts, quas, n, repeat, 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: **repeat** – 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 (e05uc) with the object supplied to nag_glopt_nlp_multistart_sqp (e05uc).

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 jth component of the ith starting point.

2: **user** – INTEGER array

3: **mode** – INTEGER

If you set **mode** to a negative value then nag_glopt_nlp_multistart_sqp (e05uc) will terminate immediately with **ifail** $= 9$.

10: **repeat** – 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.

11: **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: $1 \leq$ **nb** $\leq$ **npts**.

12: **iopts**(**740**) – INTEGER array

13: **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 (e05uc) and nag_glopt_optset (e05zk).

5.2 Optional Input Parameters

1: **nclin** – INTEGER

Default: the first dimension of the array **a**.

n_L, the number of general linear constraints.

Constraint: **nclin** ≥ 0.

2: **user** – INTEGER array

user is not used by nag_glopt_nlp_multistart_sqp (e05uc), 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**$(ldx, \mathbf{nb})$ – REAL (KIND=nag_wp) array

x(j, i) contains the final estimate of the ith solution, for $j = 1, 2, \ldots, \mathbf{n}$.

2: **objf**$(\mathbf{nb})$ – REAL (KIND=nag_wp) array

objf(i) contains the value of the objective function at the final iterate for the ith solution.

3: **objgrd**$(ldobjd, \mathbf{nb})$ – REAL (KIND=nag_wp) array

objgrd(j, i) contains the gradient of the objective function for the ith solution at the final iterate (or its finite difference approximation), for $j = 1, 2, \ldots, \mathbf{n}$.

4: **iter**$(\mathbf{nb})$ – INTEGER array

iter(i) contains the number of major iterations performed to obtain the ith solution. If less than **nb** solutions are returned then **iter**$(\mathbf{nb})$ 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.

5: **c**$(ldc, \mathbf{nb})$ – REAL (KIND=nag_wp) array

If **ncnln** > 0, **c**(j, i) contains the value of the jth nonlinear constraint function c_j at the final iterate, for the ith solution, for $j = 1, 2, \ldots,$ **ncnln**.

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

6: **cjac**$(ldcjac, sdcjac, \mathbf{nb})$ – REAL (KIND=nag_wp) array

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 kth constraint function with respect to the jth variable, for $k = 1, 2, \ldots,$ **ncnln** and $j = 1, 2, \ldots, \mathbf{n}$, for the ith solution. (See the discussion of argument **cjsl** under **confun**.)

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

7: **r**$(ldr, sdr, \mathbf{nb})$ – REAL (KIND=nag_wp) array

$sdr = \mathbf{n}$.

For each of the **nb** solutions **r** will contain a form of the Hessian; for the ith returned solution **r**(ldr, sdr, i) contains the Hessian that would be returned from the local minimizer. If **Hessian** = NO, the default, each **r**(ldr, sdr, i) contains the upper triangular Cholesky factor R of $Q^{\mathrm{T}}HQ$, an estimate of the transformed and reordered Hessian of the Lagrangian at x. If **Hessian** = YES, **r**(ldr, sdr, i) contains the upper triangular Cholesky factor R of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

8: **clamda**$(ldclda, \mathbf{nb})$ – REAL (KIND=nag_wp) array

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

9: **istate**$(listat, \mathbf{nb})$ – INTEGER array

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

istate$(\boldsymbol{j}, \boldsymbol{i})$	**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 $\mathbf{bl}(j) = \mathbf{bu}(j)$.

10: **iopts**(**740**) – INTEGER array

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

12: **user** – INTEGER array

13: **info**(**nb**) – INTEGER array

info(i) contains one of 0, 1 or 6. Please see the description of each corresponding value of **ifail** on exit from nag_opt_nlp1_solve (e04uc) for detailed explanations of these exit values. As usual 0 denotes success.

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 (e05uc) 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 constraints are violated.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ifail $= 2$

No solution obtained. Linear constraints may be infeasible.

nag_glopt_nlp_multistart_sqp (e05uc) 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 $\sqrt{\epsilon}$, where ϵ 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}.*

ifail $= 3$

nag_glopt_nlp_multistart_sqp (e05uc) 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 $= 4$

No solution obtained. Many potential solutions reach iteration limit.

The ***Iteration Limit*** *may be changed using nag_glopt_optset (e05zk).*

ifail $= 7$

User-supplied derivatives probably wrong.

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 $\langle value \rangle$ 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** $= \langle value \rangle$.

*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 (e05uc), 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 an argument for **start** (see the description of **start**) and **repeat** $=$ *false* then a randomly chosen value for **iskip** is used, otherwise **iskip** is set to 100. If **repeat** 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 **repeat** should be set to *true*.

8.1 Description of the Printed Output

See Section 9.1 in nag_opt_nlp1_solve (e04uc).

9 Example

This example finds the global minimum of the two-dimensional Schwefel function:

$$\underset{\mathbf{x}\in R^2}{\text{minimize}} f = \sum_{j=1}^{2} x_j \sin\left(\sqrt{|x_j|}\right)$$

subject to the constraints:

$$\begin{gathered} -10000 < 3.0x_1 - 2.0x_2 < 10.0, \\ -1.0 < x_1^2 - x_2^2 + 3.0x_1x_2 < 500000.0, \\ -0.9 < \cos\left((x_1/200)^2 + (x_2/100)\right) < 0.9, \\ -500 \leq x_1 \leq 500, \\ -500 \leq x_2 \leq 500. \end{gathered}$$

9.1 Program Text

```
     function e05uc_example

fprintf('e05uc example results\n\n');

npts   = nag_int(1000);
repeat = true;
nclin  = nag_int(1);
ncnln  = nag_int(2);
nb     = nag_int(10);
n      = nag_int(2);

bl = [-500; -500; -10000; -1;      -0.9];
bu = [ 500;  500;     10;  500000;  0.9];
a  = [3, -2];

% Initialise e05uc
iopts = zeros(740, 1, nag_int_name);
opts  = zeros(485, 1);
[iopts, opts, ifail] = e05zk(...
                             'Initialize = e05uc', iopts, opts);
[iopts, opts, ifail] = e05zk(...
                             'Derivative Level = 3', iopts, opts);

wstat = warning();
warning('OFF');
% Solve the problem
[x, objf, objgrd, iter, c, cjac, r, clamda, istate, ...
                          iopts, opts, user, info, ifail] = ...
  e05uc(...
        n, ncnln, a, bl, bu, @confun, @objfun, ...
        npts, 'e05ucz', repeat, nb, iopts, opts);

warning(wstat);

if ifail == 8
  l = double(info(10));
  fprintf('\nOnly %d solutions found\n', l);
else
  l = 10;
end

l = min(l,3);
% List details of first 3 solutions only
for i=1:l
  fprintf('\nSolution number %d\n\n', i);
  fprintf('e04uc returned with ifail = %d\n\n', info(i));
  fprintf(' Variable Istate           Value Lagrange Multiplier\n');
  for j=1:double(n)
    fprintf(' %3d%9d%19.6g%13.4g\n', j, istate(j,i), x(j,i), clamda(j,i));
  end

  if nclin > 0
    ax = a*x(:,i);
    fprintf('\n L Con    Istate           Value Lagrange Multiplier\n');
    for k=double(n+1):double(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=double(n+nclin+1):double(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, objf, objgrd, user] = objfun(mode, n, x, objgrd, nstate, user)
  if mode==0 || mode==2
    % Evaluate the objective function.
    objf = x(1)*sin(sqrt(abs(x(1)))) + x(2)*sin(sqrt(abs(x(2))));
  else
    objf = 0;
  end

  if mode==1 || mode==2
    % Calculate the gradient of the objective function.
    t = sqrt(abs(x(1)));
    objgrd(1) = sin(t) + 0.5*t*cos(t);
    t = sqrt(abs(x(2)));
    objgrd(2) = sin(t) + 0.5*t*cos(t);
  end

function [mode, c, cjsl, user] = ...
    confun(mode, ncnln, n, ldcj1, needc, x, cjsl, nstate, user)
  c = zeros(ncnln, 1);
  dncnln = double(ncnln);

  if (mode==0 || mode==2)
    % Constraint values are required.
    % Only those for which needc is non-zero need be set.
    for k = 1:dncnln
      if (needc(k)>0)
        switch k
          case {1}
            c(k) = x(1)^2 - x(2)^2 + 3.0*x(1)*x(2);
          case {2}
            c(k) = cos((x(1)/200.0)^2+(x(2)/100.0));
        end
      end
    end
  end

  if (mode==1 || mode==2)
    % Constraint derivatives (cjsl) are required.
    for k = 1:dncnln
      switch k
        case {1}
          cjsl(k,1) = 2.0*x(1) + 3.0*x(2);
          cjsl(k,2) = -2.0*x(2) + 3.0*x(1);
        case {2}
          t1 = x(1)/200.0;
          t2 = x(2)/100.0;
          cjsl(k,1) = -sin(t1^2+t2)*2.0*t1/200.0;
          cjsl(k,2) = -sin(t1^2+t2)/100.0;
      end
    end
  end
```

9.2 Program Results

```
     e05uc example results

Solution number 1

e04uc returned with ifail = 0

 Variable Istate          Value Lagrange Multiplier
   1        0           -394.151            0
   2        0           -433.491            0

 L Con    Istate          Value Lagrange Multiplier
```

```
   1        0           -315.472            0

 NL Con  Istate            Value Lagrange Multiplier
   1        0             480024            0
   2        2                0.9            0

Final objective value =       -731.7064
clamda:          0         0         0         0 -718.9449

 -----------------------------------------------------

Solution number 2

e04uc returned with ifail = 0

 Variable Istate           Value Lagrange Multiplier
   1        0           -413.805            0
   2        0           -382.984            0

 L Con    Istate           Value Lagrange Multiplier
   1        0           -475.447            0

 NL Con  Istate            Value Lagrange Multiplier
   1        2             500000            0
   2        2                0.9            0

Final objective value =       -665.1962
clamda:    1.0e+03 *

         0         0         0   -0.0000   -1.1615

 -----------------------------------------------------

Solution number 3

e04uc returned with ifail = 1

 Variable Istate           Value Lagrange Multiplier
   1        0           -413.964            0
   2        0           -382.327            0

 L Con    Istate           Value Lagrange Multiplier
   1        0           -477.237            0

 NL Con  Istate            Value Lagrange Multiplier
   1        2             500000            0
   2        0           0.895662            0

Final objective value =       -660.1803
clamda:          0         0         0   -0.0018         0

 -----------------------------------------------------
```

10 Algorithmic Details

See Section 11 in nag_opt_nlp1_solve (e04uc).

11 Optional Parameters

Several optional parameters in nag_glopt_nlp_multistart_sqp (e05uc) 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 (e05uc) 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 (e05uc). Before calling nag_glopt_nlp_multistart_sqp (e05uc), the

optional parameter arrays **iopts** and **opts must** be initialized for use with nag_glopt_nlp_multistart_sqp (e05uc) 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 (e05uc) (unless they define invalid values) and so remain in effect for subsequent calls to nag_glopt_nlp_multistart_sqp (e05uc).

11.1 Description of the Optional Parameters

nag_glopt_nlp_multistart_sqp (e05uc) supports two options that are distinct from those of nag_opt_nlp1_solve (e04uc):

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 (e05uc).

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	**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.

See Section 12 in nag_opt_nlp1_solve (e04uc) for details of the other options.

The **Warm Start** option of nag_opt_nlp1_solve (e04uc) is not a valid option for use with nag_glopt_nlp_multistart_sqp (e05uc).

12 Description of Monitoring Information

See Section 13 in nag_opt_nlp1_solve (e04uc).