NAG Library Function Document

nag_glopt_nlp_pso (e05sbc)

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

1 Purpose

nag_glopt_nlp_pso (e05sbc) is designed to search for the global minimum or maximum of an arbitrary function, subject to general nonlinear constraints, using Particle Swarm Optimization (PSO). Derivatives are not required, although these may be used by an accompanying local minimization function if desired. nag_glopt_nlp_pso (e05sbc) is essentially identical to nag_glopt_bnd_pso (e05sac), with an expert interface and various additional arguments added; otherwise most arguments are identical. In particular, nag_glopt_bnd_pso (e05sac) does not handle general constraints.

2 Specification

```
#include <nag.h>
#include <nage05.h>

void nag_glopt_nlp_pso (Integer ndim, Integer ncon, Integer npar,
     double xb[], double *fb, double cb[], const double bl[],
     const double bu[], double xbest[], double fbest[], double cbest[],

     void (*objfun)(Integer *mode, Integer ndim, const double x[],
          double *objf, double vecout[], Integer nstate, Nag_Comm *comm),

     void (*confun)(Integer *mode, Integer ncon, Integer ndim, Integer tdcj,
          const Integer needc[], const double x[], double c[], double cjac[],
          Integer nstate, Nag_Comm *comm),

     void (*monmod)(Integer ndim, Integer ncon, Integer npar, double x[],
          const double xb[], double fb, const double cb[],
          const double xbest[], const double fbest[], const double cbest[],
          const Integer itt[], Nag_Comm *comm, Integer *inform),

     Integer iopts[], double opts[], Nag_Comm *comm, Integer itt[],
     Integer *inform, NagError *fail)
```

Before calling nag_glopt_nlp_pso (e05sbc), nag_glopt_opt_set (e05zkc) **must** be called with **optstr** set to '`Initialize = e05sbc`'. Optional arguments may also be specified by calling nag_glopt_opt_set (e05zkc) before the call to nag_glopt_nlp_pso (e05sbc).

3 Description

nag_glopt_nlp_pso (e05sbc) uses a stochastic method based on Particle Swarm Optimization (PSO) to search for the global optimum of a nonlinear function F, subject to a set of bound constraints on the variables, and optionally a set of general nonlinear constraints. In the PSO algorithm (see Section 11), a set of particles is generated in the search space, and advances each iteration to (hopefully) better positions using a heuristic velocity based upon *inertia*, *cognitive memory* and *global memory*. The inertia is provided by a decreasingly weighted contribution from a particles current velocity, the cognitive memory refers to the best candidate found by an individual particle and the global memory refers to the best candidate found by all the particles. This allows for a global search of the domain in question.

Further, this may be coupled with a selection of local minimization functions, which may be called during the iterations of the heuristic algorithm, the *interior* phase, to hasten the discovery of locally optimal points, and after the heuristic phase has completed to attempt to refine the final solution, the *exterior* phase. Different options may be set for the local optimizer in each phase.

Without loss of generality, the problem is assumed to be stated in the following form:

$$\underset{\mathbf{x}\in R^{ndim}}{\text{minimize}}\, F(\mathbf{x}) \quad \text{subject to} \quad \boldsymbol{\ell} \leq \begin{pmatrix} \mathbf{x} \\ \mathbf{c}(\mathbf{x}) \end{pmatrix} \leq \mathbf{u},$$

where the objective $F(\mathbf{x})$ is a scalar function, $\mathbf{c}(\mathbf{x})$ is a vector of scalar constraint functions, $\mathbf{x}$ is a vector in R^{ndim} and the vectors $\boldsymbol{\ell} \leq \mathbf{u}$ are lower and upper bounds respectively for the $ndim$ variables and $ncon$ constraints. Both the objective function and the $ncon$ constraints may be nonlinear. Continuity of F, and the functions $\mathbf{c}(\mathbf{x})$, is not essential. For functions which are smooth and primarily unimodal, faster solutions will almost certainly be achieved by using Chapter e04 functions directly.

For functions which are smooth and multi-modal, gradient dependent local minimization functions may be coupled with nag_glopt_nlp_pso (e05sbc).

For multi-modal functions for which derivatives cannot be provided, particularly functions with a significant level of noise in their evaluation, nag_glopt_nlp_pso (e05sbc) should be used either alone, or coupled with nag_opt_simplex_easy (e04cbc).

For heavily constrained problems, nag_glopt_nlp_pso (e05sbc) should either be used alone, or coupled with nag_opt_nlp (e04ucc) provided the function and the constraints are sufficiently smooth.

The $ndim$ lower and upper box bounds on the variable $\mathbf{x}$ are included to initialize the particle swarm into a finite hypervolume, although their subsequent influence on the algorithm is user determinable (see the option **Boundary** in Section 12). It is strongly recommended that sensible bounds are provided for all variables and constraints.

nag_glopt_nlp_pso (e05sbc) may also be used to maximize the objective function, or to search for a feasible point satisfying the simple bounds and general constraints (see the option **Optimize**).

Due to the nature of global optimization, unless a predefined target is provided, there is no definitive way of knowing when to end a computation. As such several stopping heuristics have been implemented into the algorithm. If any of these is achieved, nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** = NW_SOLUTION_NOT_GUARANTEED, and the parameter **inform** will indicate which criteria was reached. See **inform** for more information.

In addition, you may provide your own stopping criteria through **monmod**, **objfun** and **confun**.

nag_glopt_bnd_pso (e05sac) provides a simpler interface, without the inclusion of general nonlinear constraints.

4 References

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

Kennedy J and Eberhart R C (1995) Particle Swarm Optimization *Proceedings of the 1995 IEEE International Conference on Neural Networks* 1942–1948

Koh B, George A D, Haftka R T and Fregly B J (2006) Parallel Asynchronous Particle Swarm Optimization *International Journal for Numerical Methods in Engineering* **67(4)** 578–595

Vaz A I and Vicente L N (2007) A Particle Swarm Pattern Search Method for Bound Constrained Global Optimization *Journal of Global Optimization* **39(2)** 197–219 Kluwer Academic Publishers

5 Arguments

Note: for descriptions of the symbolic variables, see Section 11.

1: **ndim** – Integer *Input*

On entry: $ndim$, the number of dimensions.

Constraint: **ndim** ≥ 1.

2: **ncon** – Integer *Input*

On entry: $ncon$, the number of constraints, not including box constraints.

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

3: **npar** – Integer *Input*

On entry: $npar$, the number of particles to be used in the swarm. Assuming all particles remain within constraints, each complete iteration will perform at least **npar** function evaluations. Otherwise, significantly fewer objective function evaluations may be performed.

Suggested value: $\mathbf{npar} = 10 \times \mathbf{ndim}$.

Constraint: $\mathbf{npar} \geq 5$.

4: **xb**[**ndim**] – double *Output*

On exit: the location of the best solution found, $\tilde{\mathbf{x}}$, in R^{ndim}.

5: **fb** – double * *Output*

On exit: the objective value of the best solution, $\tilde{f} = F(\tilde{\mathbf{x}})$.

6: **cb**[**ncon**] – double *Output*

On exit: the constraint violations of the best solution found, $\tilde{\mathbf{e}} = \mathbf{e}(\tilde{\mathbf{x}})$. These may have been deemed to be acceptable given the tolerance and scaling of the constraints. See Sections 11 and 12.

7: **bl**[**ndim** + **ncon**] – const double *Input*
8: **bu**[**ndim** + **ncon**] – const double *Input*

On entry: **bl** is $\boldsymbol{\ell}$, the array of lower bounds, **bu** is $\mathbf{u}$, the array of upper bounds. The first **ndim** entries in **bl** and **bu** must contain the lower and upper simple (box) bounds of the variables respectively. These must be provided to initialize the sample population into a finite hypervolume, although their subsequent influence on the algorithm is user determinable (see the option **Boundary** in Section 12).

The next **ncon** entries must contain the lower and upper bounds for any general constraints respectively.

If $\mathbf{bl}[i-1] = \mathbf{bu}[i-1]$ for any $i \in \{1, \ldots, \mathbf{ndim}\}$, variable i will remain locked to $\mathbf{bl}[i-1]$ regardless of the **Boundary** option selected.

It is strongly advised that you place sensible lower and upper bounds on all variables and constraints, even if your model allows for unbounded variables or constraints.

Constraints:

$\mathbf{bl}[i-1] \leq \mathbf{bu}[i-1]$, for $i = 1, 2, \ldots, \mathbf{ndim} + \mathbf{ncon}$;
$\mathbf{bl}[i-1] \neq \mathbf{bu}[i-1]$ for at least one $i \in \{1, \ldots, \mathbf{ndim}\}$.

9: **xbest**[**ndim** × **npar**] – double *Input/Output*

Note: the ith component of the best position of the jth particle, $\hat{x}_j(i)$, is stored in $\mathbf{xbest}[(j-1) \times \mathbf{ndim} + i - 1]$.

On entry: if using **Start** = WARM, the initial particle positions, $\hat{\mathbf{x}}_j^0$.

On exit: the best positions found, $\hat{\mathbf{x}}_j$, by the **npar** particles in the swarm.

10: **fbest**[**npar**] – double *Input/Output*

On entry: if using **Start** = WARM, objective function values, $\hat{f}_j^0 = F\left(\hat{\mathbf{x}}_j^0\right)$, corresponding to the **npar** particle locations stored in **xbest**.

On exit: objective function values, $\hat{f}_j = F(\hat{\mathbf{x}}_j)$, corresponding to the locations returned in **xbest**.

11: **cbest**[**ncon** × **npar**] – double *Input/Output*

Note: the kth constraint violation of the jth particle is stored in **cbest**$[(j-1) \times$ **ncon** $+ k - 1]$.

On entry: if using **Start** = WARM, the initial constraint violations, $\hat{\mathbf{e}}_j^0 = \mathbf{e}\left(\hat{\mathbf{x}}_j^0\right)$, corresponding to the **npar** particle locations.

On exit: the final constraint violations, $\hat{\mathbf{e}}_j$, corresponding to the locations returned in **xbest**.

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

objfun must, depending on the value of **mode**, calculate the objective function *and/or* calculate the gradient of the objective function for a $ndim$-variable vector **x**. Gradients are only required if a local minimizer has been chosen which requires gradients. See the option **Local Minimizer** for more information.

The specification of **objfun** is:

```
void objfun (Integer *mode, Integer ndim, const double x[],
     double *objf, double vecout[], Integer nstate, Nag_Comm *comm)
```

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

On entry: indicates which functionality is required.

mode = 0
$F(\mathbf{x})$ should be returned in **objf**. The value of **objf** on entry may be used as an upper bound for the calculation. Any expected value of $F(\mathbf{x})$ that is greater than **objf** may be approximated by this upper bound; that is **objf** can remain unaltered.

mode = 1
Local Minimizer = e04ucc only
First derivatives can be evaluated and returned in **vecout**. Any unaltered elements of **vecout** will be approximated using finite differences.

mode = 2
Local Minimizer = e04ucc only
$F(\mathbf{x})$ *must* be calculated and returned in **objf**, and available first derivatives can be evaluated and returned in **vecout**. Any unaltered elements of **vecout** will be approximated using finite differences.

mode = 5
$F(\mathbf{x})$ *must* be calculated and returned in **objf**. The value of **objf** on entry may not be used as an upper bound.

mode = 6
Local Minimizer = e04dgc only
All first derivatives *must* be evaluated and returned in **vecout**.

mode = 7
Local Minimizer = e04dgc only
$F(\mathbf{x})$ *must* be calculated and returned in **objf**, and *all* first derivatives *must* be evaluated and returned in **vecout**.

On exit: if the value of **mode** is set to be negative, then nag_glopt_nlp_pso (e05sbc) will exit as soon as possible with **fail.code** = NE_USER_STOP and **inform** = **mode**.

2: **ndim** – Integer *Input*

On entry: the number of dimensions.

3: **x**[**ndim**] – const double *Input*

On entry: **x**, the point at which the objective function and/or its gradient are to be evaluated.

4: **objf** – double * *Input/Output*

On entry: the value of **objf** passed to **objfun** varies with the argument **mode**.

mode $= 0$
: **objf** is an upper bound for the value of $F(\mathbf{x})$, often equal to the best constraint penalised value of $F(\mathbf{x})$ found so far by a given particle if the objective function is strictly positive (see Section 11). Only objective function values less than the value of **objf** on entry will be used further. As such this upper bound may be used to stop further evaluation when this will only increase the objective function value above the upper bound.

mode $= 1$, 2, 5, 6 or 7
: **objf** is meaningless on entry.

On exit: the value of **objf** returned varies with the argument **mode**.

mode $= 0$
: **objf** must be the value of $F(\mathbf{x})$. Only values of $F(\mathbf{x})$ strictly less than **objf** on entry need be accurate.

mode $= 1$ or 6
: Need not be set.

mode $= 2$, 5 or 7
: $F(\mathbf{x})$ must be calculated and returned in **objf**. The entry value of **objf** may not be used as an upper bound.

5: **vecout**[**ndim**] – double *Input/Output*

On entry: if **Local Minimizer** $=$ e04ucc, the values of **vecout** are used internally to indicate whether a finite difference approximation is required. See nag_opt_nlp (e04ucc).

On exit: the required values of **vecout** returned to the calling function depend on the value of **mode**.

mode $= 0$ or 5
: The value of **vecout** need not be set.

mode $= 1$ or 2
: **vecout** can contain components of the gradient of the objective function $\frac{\partial F}{\partial x_i}$ for some $i = 1, 2, \ldots$ **ndim**, or acceptable approximations. Any unaltered elements of **vecout** will be approximated using finite differences.

mode $= 6$ or 7
: **vecout** must contain the gradient of the objective function $\frac{\partial F}{\partial x_i}$ for all $i = 1, 2, \ldots$ **ndim**. Approximation of the gradient is strongly discouraged, and no finite difference approximations will be performed internally (see nag_opt_conj_grad (e04dgc)).

6: **nstate** – Integer *Input*

On entry: **nstate** indicates various stages of initialization throughout the function. This allows for permanent global arguments to be initialized the least number of times. For example, you may initialize a random number generator seed.

nstate $= 2$
: **objfun** is called for the very first time. You may save computational time if certain data must be read or calculated only once.

nstate $= 1$
: **objfun** is called for the first time by a NAG local minimization function. You may save computational time if certain data required for the local minimizer need only be calculated at the initial point of the local minimization.

nstate $= 0$
: Used in all other cases.

7: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_glopt_nlp_pso (e05sbc) you may allocate memory and initialize these pointers with various quantities for use by **objfun** when called from nag_glopt_nlp_pso (e05sbc) (see Section 3.2.1.1 in the Essential Introduction).

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

confun must calculate any constraints other than the box constraints. If no constraints are required, **confun** may be **NULLFN**. For information on how a NAG local minimizer will use **confun** see the documentation for nag_opt_nlp (e04ucc).

The specification of **confun** is:

```
void confun (Integer *mode, Integer ncon, Integer ndim, Integer tdcj,
     const Integer needc[], const double x[], double c[],
     double cjac[], Integer nstate, Nag_Comm *comm)
```

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

On entry: indicates which values must be assigned during each call of **confun**. Only the following values need be assigned, for each value of $k \in \{1, \ldots, \mathbf{ncon}\}$ such that $\mathbf{needc}[k-1] > 0$:

mode $= 0$
: the constraint values $c_k(\mathbf{x})$.

mode $= 1$
: rows of the constraint jacobian, $\frac{\partial c_k}{\partial x_i}(\mathbf{x})$, for $i = 1, 2, \ldots, \mathbf{ndim}$.

mode $= 2$
: the constraint values $c_k(\mathbf{x})$ and the corresponding rows of the constraint jacobian, $\frac{\partial c_k}{\partial x_i}(\mathbf{x})$, for $i = 1, 2, \ldots, \mathbf{ndim}$.

On exit: may be set to a negative value if you wish to terminate the solution to the current problem. In this case nag_glopt_nlp_pso (e05sbc) will terminate with **fail.code** $=$ NE_USER_STOP and **inform** $=$ **mode** as soon as possible.

2: **ncon** – Integer *Input*

On entry: the number of constraints, not including box bounds.

3: **ndim** – Integer *Input*

On entry: the number of variables.

4: **tdcj** – Integer *Input*

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

5: **needc**[**ncon**] – const Integer *Input*

On entry: the indices of the elements of **c** and/or **cjac** that must be evaluated by **confun**. If **needc**$[k-1] > 0$, the $k-1$th element of **c**, corresponding to the values of the kth constraint, and/or the available elements of the $k-1$th row of **cjac**, corresponding to the derivatives of the kth constraint, must be evaluated at **x** (see argument **mode**).

6: **x**[**ndim**] – const double *Input*

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

7: **c**[**ncon**] – double *Output*

On exit: if **needc**$[k-1] > 0$ and **mode** $= 0$ or 2, **c**$[k-1]$ must contain the value of $c_k(\mathbf{x})$. The remaining elements of **c**, corresponding to the non-positive elements of **needc**, need not be set.

8: **cjac**[**ncon** $\times$ **tdcj**] – double *Input/Output*

Note: the dimension, *dim*, of the array **cjac** is **tdcj** $\times$ **ndim**.

Note: the derivative of the kth constraint with respect to the ith component, $\frac{\partial c_k}{\partial x_{\mathrm{i}}}$, is stored in **cjac**$[(k-1) \times$ **tdcj** $+ i - 1]$, which we denote as **CJAC**(k, i).

On entry: the elements of **cjac** are set to special values which enable nag_glopt_nlp_pso (e05sbc) to detect whether they are changed by **confun**.

On exit: if **needc**$[k-1] > 0$ and **mode** $= 1$ or 2, the elements of **cjac** corresponding to the kth row of the constraint jacobian should contain the available elements of the vector ∇c_k given by

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

where $\frac{\partial c_k}{\partial x_i}$ is the partial derivative of the kth constraint with respect to the ith variable, evaluated at the point **x**; elements of **cjac** that remain unaltered will be approximated internally using finite differences. The remaining rows of **cjac**, corresponding to non-positive elements of **needc**, need not be set.

It must be emphasized that unassigned elements of **cjac** are not treated as constant; they are estimated by finite differences, at nontrivial expense. An interval for each element of **x** is computed automatically at the start of the optimization. The automatic procedure can usually identify constant elements of **cjac**, which are then computed once only by finite differences.

9: **nstate** – Integer *Input*

On entry: **nstate** indicates various stages of initialization throughout the function. This allows for permanent global arguments to be initialized a minimum number of times. For example, you may initialize a random number generator seed. Note that unless the option **Optimize** $=$ CONSTRAINTS has been set, **objfun** will be called before **confun**.

nstate $= 2$

confun is called for the very first time. This argument setting allows you to save computational time if certain data must be read or calculated only once.

nstate $= 1$

confun is called for the first time during a NAG local minimization function. This argument setting allows you to save computational time if certain data required for the local minimizer need only be calculated at the initial point of the local minimization.

nstate $= 0$
Used in all other cases.

10: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_glopt_nlp_pso (e05sbc) you may allocate memory and initialize these pointers with various quantities for use by **confun** when called from nag_glopt_nlp_pso (e05sbc) (see Section 3.2.1.1 in the Essential Introduction).

confun should be tested separately before being used in conjunction with nag_glopt_nlp_pso (e05sbc).

14: **monmod** – function, supplied by the user *External Function*

A user-specified monitoring and modification function. **monmod** is called once every complete iteration after a finalization check. It may be used to modify the particle locations that will be evaluated at the next iteration. This permits the incorporation of algorithmic modifications such as including additional advection heuristics and genetic mutations. **monmod** is only called during the main loop of the algorithm, and as such will be unaware of any further improvement from the final local minimization. If no monitoring and/or modification is required, **monmod** may be NULLFN.

The specification of **monmod** is:

```
void monmod (Integer ndim, Integer ncon, Integer npar, double x[],
     const double xb[], double fb, const double cb[],
     const double xbest[], const double fbest[], const double cbest[],
     const Integer itt[], Nag_Comm *comm, Integer *inform)
```

1: **ndim** – Integer *Input*

On entry: the number of dimensions.

2: **ncon** – Integer *Input*

On entry: the number of constraints.

3: **npar** – Integer *Input*

On entry: the number of particles.

4: **x**[**ndim** × **npar**] – double *Input/Output*

Note: the ith component of the jth particle, $x_j(i)$, is stored in **x**$[(j-1)\times$ **ndim** $+ i - 1]$.

On entry: the **npar** particle locations, $\mathbf{x}_j$, which will currently be used during the next iteration unless altered in **monmod**.

On exit: the particle locations to be used during the next iteration.

5: **xb**[**ndim**] – const double *Input*

On entry: the location, $\tilde{\mathbf{x}}$, of the best solution yet found.

6: **fb** – double *Input*

On entry: the objective value, $\tilde{f} = F(\tilde{\mathbf{x}})$, of the best solution yet found.

7: **cb**[**ncon**] – const double *Input*

On entry: the constraint violations, $\tilde{\mathbf{e}} = \mathbf{e}(\tilde{\mathbf{x}})$, of the best solution yet found.

8: **xbest**[**ndim** × **npar**] – const double *Input*

Note: the ith component of the position of the jth particle's cognitive memory, $\hat{x}_j(i)$, is stored in **xbest**[$(j-1) \times$ **ndim** $+ i - 1$].

On entry: the locations currently in the cognitive memory, $\hat{\mathbf{x}}_j$, for $j = 1, 2, \ldots,$ **npar** (see Section 11).

9: **fbest**[**npar**] – const double *Input*

On entry: the objective values currently in the cognitive memory, $F(\hat{\mathbf{x}}_j)$, for $j = 1, 2, \ldots,$ **npar**.

10: **cbest**[**ncon** × **npar**] – const double *Input*

Note: the kth constraint violation of the jth particle's cognitive memory is stored in **cbest**[$(j-1) \times$ **ncon** $+ k - 1$].

On entry: the constraint violations currently in the cognitive memory, $\hat{\mathbf{e}} = \mathbf{e}(\hat{\mathbf{x}}_j)$, for $j = 1, 2, \ldots,$ **npar**, evaluated at $\hat{\mathbf{x}}_j$.

11: **itt**[7] – const Integer *Input*

On entry: iteration and function evaluation counters (see description of **itt** below).

12: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_glopt_nlp_pso (e05sbc) you may allocate memory and initialize these pointers with various quantities for use by **monmod** when called from nag_glopt_nlp_pso (e05sbc) (see Section 3.2.1.1 in the Essential Introduction).

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

On entry: **inform** $= 0$

On exit: setting **inform** < 0 will cause near immediate exit from nag_glopt_nlp_pso (e05sbc). This value will be returned as **inform** with **fail.code** = NE_USER_STOP. You need not set **inform** unless you wish to force an exit.

15: **iopts**[*dim*] – Integer *Communication Array*

Note: the dimension, *dim*, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **iopts** in the previous call to nag_glopt_opt_set (e05zkc).

On entry: optional argument array as generated and possibly modified by calls to nag_glopt_opt_set (e05zkc). The contents of **iopts** MUST NOT be modified directly between calls to nag_glopt_nlp_pso (e05sbc), nag_glopt_opt_set (e05zkc) or nag_glopt_opt_get (e05zlc).

16: **opts**[*dim*] – double *Communication Array*

Note: the dimension, *dim*, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **opts** in the previous call to nag_glopt_opt_set (e05zkc).

On entry: optional argument array as generated and possibly modified by calls to nag_glopt_opt_set (e05zkc). The contents of **opts** MUST NOT be modified directly between calls to nag_glopt_nlp_pso (e05sbc), nag_glopt_opt_set (e05zkc) or nag_glopt_opt_get (e05zlc).

17: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

18: **itt**[**7**] – Integer *Output*

On exit: integer iteration counters for nag_glopt_nlp_pso (e05sbc).

itt[0]
Number of complete iterations.

itt[1]
Number of complete iterations without improvement to the current optimum.

itt[2]
Number of particles converged to the current optimum.

itt[3]
Number of improvements to the optimum.

itt[4]
Number of function evaluations performed.

itt[5]
Number of particles reset.

itt[6]
Number of violated constraints at completion. Note this is always calculated using the L^1 norm and a nonzero result does not necessarily mean that the algorithm did not find a suitably constrained point with respect to the single norm used.

19: **inform** – Integer * *Output*

On exit: indicates which finalization criterion was reached. The possible values of **inform** are:

inform	**Meaning**
< 0	Exit from a user-supplied subroutine.
0	nag_glopt_nlp_pso (e05sbc) has detected an error and terminated.
1	The provided objective target has been achieved. (**Target Objective Value**).
2	The standard deviation of the location of all the particles is below the set threshold (**Swarm Standard Deviation**). If the solution returned is not satisfactory, you may try setting a smaller value of **Swarm Standard Deviation**, or try adjusting the options governing the repulsive phase (**Repulsion Initialize**, **Repulsion Finalize**).
3	The total number of particles converged (**Maximum Particles Converged**) to the current global optimum has reached the set limit. This is the number of particles which have moved to a distance less than **Distance Tolerance** from the optimum with regard to the L^2 norm. If the solution is not satisfactory, you may consider lowering the **Distance Tolerance**. However, this may hinder the global search capability of the algorithm.

4	The maximum number of iterations without improvement (**Maximum Iterations Static**) has been reached, and the required number of particles (**Maximum Iterations Static Particles**) have converged to the current optimum. Increasing either of these options will allow the algorithm to continue searching for longer. Alternatively if the solution is not satisfactory, re-starting the application several times with **Repeatability** = OFF may lead to an improved solution.
5	The maximum number of iterations (**Maximum Iterations Completed**) has been reached. If the number of iterations since improvement is small, then a better solution may be found by increasing this limit, or by using the option **Local Minimizer** with corresponding exterior options. Otherwise if the solution is not satisfactory, you may try re-running the application several times with **Repeatability** = OFF and a lower iteration limit, or adjusting the options governing the repulsive phase (**Repulsion Initialize**, **Repulsion Finalize**).
6	The maximum allowed number of function evaluations (**Maximum Function Evaluations**) has been reached. As with **inform** = 5, increasing this limit if the number of iterations without improvement is small, or decreasing this limit and running the algorithm multiple times with **Repeatability** = OFF, may provide a superior result.
7	A feasible point has been found. The objective has not been minimized, although it has been evaluated at the final solutions given in **xb** and **xbest** (**Optimize** = CONSTRAINTS).

If you wish to continue from the final position gained from a previous simulation with adjusted options, you may set the option **Start** = WARM, and pass back in the returned arrays **xbest**, **fbest**, and **cbest**. You should either record the returned values of **xb**, **fb** and **cb** for comparison, as these will not be re-used by the algorithm, or include them in **xbest**, **fbest** and **cbest** respectively by overwriting the entries corresponding to one particle with the relevant information.

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

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

nag_glopt_nlp_pso (e05sbc) returns **fail**.**code** = NE_NOERROR if and only if a finalization criterion has been reached which can guarantee success. This may only happen if:

These finalization criteria are not active using default option settings, and must be explicitly set using nag_glopt_opt_set (e05zkc) if required.

nag_glopt_nlp_pso (e05sbc) will return **fail**.**code** = NW_SOLUTION_NOT_GUARANTEED if no error has been detected, and a finalization criterion has been achieved which cannot guarantee success. This does not indicate that the function has failed, merely that the returned solution cannot be guaranteed to be the true global optimum.

The value of **inform** should be examined to determine which finalization criterion was reached.

6 Error Indicators and Warnings

NE_ALLOC_FAIL

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

NE_BAD_PARAM

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

NE_BOUND

On entry, **bl**[i] = **bu**[i] for all box bounds i.
Constraint: **bu**[i] > **bl**[i] for at least one box bound i.

On entry, **bl**[⟨*value*⟩] = ⟨*value*⟩ and **bu**[⟨*value*⟩] = ⟨*value*⟩.
Constraint: **bu**[i] ≥ **bl**[i] for all i.

NE_DERIV_ERRORS

Derivative checks indicate possible errors in the supplied derivatives. Gradient checks may be disabled by setting **Verify Gradients** = OFF.

NE_ILLEGAL_CALLBACK

nag_glopt_nlp_pso (e05sbc) has been called with **ncon** > 0 and **confun** is **NULL**. Only use **NULL** with **ncon** = 0.

NE_INT

On entry, **ncon** = ⟨*value*⟩.
Constraint: **ncon** ≥ 0.

On entry, **ndim** = ⟨*value*⟩.
Constraint: **ndim** ≥ 1.

On entry, **npar** = ⟨*value*⟩.
Constraint: **npar** ≥ 5 × **num_threads**, where **num_threads** is the value returned by the OpenMP environment variable `OMP_NUM_THREADS`, or **num_threads** is 1 for a serial version of this function.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_INVALID_OPTION

Either the option arrays have not been initialized for nag_glopt_nlp_pso (e05sbc), or they have become corrupted.

The option **Optimize** = CONSTRAINTS is active, however **ncon** = 0.

NE_NO_LICENCE

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

NE_USER_STOP

User requested exit ⟨*value*⟩ during call to **confun**.

User requested exit ⟨*value*⟩ during call to **monmod**.

User requested exit ⟨*value*⟩ during call to **objfun**.

NW_FAST_SOLUTION

If the option **Target Warning** has been activated, this indicates that the **Target Objective Value** has been achieved to specified tolerances at a sufficiently constrained point, either during the initialization phase, or during the first two iterations of the algorithm. While this is not necessarily an error, it may occur if:

(i) The target was achieved at the first point sampled by the function. This will be the mean of the lower and upper bounds.

(ii) The target may have been achieved at a randomly generated sample point. This will always be a possibility provided that the domain under investigation contains a point with a target objective value.

(iii) If the **Local Minimizer** has been set, then a sample point may have been inside the basin of attraction of a satisfactory point. If this occurs repeatedly when the function is called, it may imply that the objective is largely unimodal, and that it may be more efficient to use the function selected as the **Local Minimizer** directly.

Assuming that **objfun** is correct, you may wish to set a better **Target Objective Value**, or a stricter **Target Objective Tolerance**.

NW_NOT_FEASIBLE

Unable to locate strictly feasible point. $\langle value \rangle$ constraints remain violated. This exit may be suppressed using the option **Constraint Warning**.

NW_SOLUTION_NOT_GUARANTEED

A finalization criterion was reached that cannot guarantee success.
On exit, **inform** $= \langle value \rangle$.

7 Accuracy

If **fail.code** = NE_NOERROR (or **fail.code** = NW_FAST_SOLUTION) or **fail.code** = NW_SOLUTION_NOT_GUARANTEED on exit, a criterion will have been reached depending on user selected options. As with all global optimization software, the solution achieved may not be the true global optimum. Various options allow for either greater search diversity or faster convergence to a (local) optimum (See Sections 11 and 12).

Provided the objective function and constraints are sufficiently well behaved, if a local minimizer is used in conjunction with nag_glopt_nlp_pso (e05sbc), then it is more likely that the final result will at least be in the near vicinity of a local optimum, and due to the global search characteristics of the particle swarm, this solution should be superior to many other local optima.

Caution should be used in accelerating the rate of convergence, as with faster convergence, less of the domain will remain searchable by the swarm, making it increasingly difficult for the algorithm to detect the basins of attraction of superior local optima. Using the options **Repulsion Initialize** and **Repulsion Finalize** described in Section 12 will help to overcome this, by causing the swarm to diverge away from the current optimum once no more local improvement is likely.

On successful exit with guaranteed success, **fail.code** = NE_NOERROR (or **fail.code** = NW_FAST_SOLUTION). This may happen if a **Target Objective Value** is assigned and is reached by the algorithm at a satisfactorily constrained point. It will also occur if a constrained point is found when **Optimize** = CONSTRAINTS is set.

On successful exit without guaranteed success, **fail.code** = NW_SOLUTION_NOT_GUARANTEED is returned. This will happen if another finalization criterion is achieved without the detection of an error.

In both cases, the value of **inform** provides further information as to the cause of the exit.

8 Parallelism and Performance

The code for nag_glopt_nlp_pso (e05sbc) is not directly threaded for parallel execution. In particular, none of the user-supplied functions will be called from within a parallel region generated by nag_glopt_nlp_pso (e05sbc).

9 Further Comments

The memory used by nag_glopt_nlp_pso (e05sbc) is relatively static throughout. Indeed, most of the memory required is used to store the current particle locations, the cognitive particle memories, the particle velocities and the particle weights. As such, nag_glopt_nlp_pso (e05sbc) may be used in problems with high dimension number (**ndim** > 100) without the concern of computational resource exhaustion, although the probability of successfully locating the global optimum will decrease dramatically with the increase in dimensionality.

Due to the stochastic nature of the algorithm, the result will vary over multiple runs. This is particularly true if arguments and options are chosen to accelerate convergence at the expense of the global search. However, the option **Repeatability** $=$ ON may be set to initialize the internal random number generator using a preset seed, which will result in identical solutions being obtained.

10 Example

This example uses a particle swarm to find 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:

$$\begin{gathered} 3.0x_1 - 2.0x_2 < 10.0, \\ -1.0 < x_1^2 - x_2^2 + 3.0x_1x_2 < 50000.0, \\ -0.9 < \cos\left((x_1/200)^2 + (x_2/100)\right) < 0.9, \\ -500 \le x_1 \le 500, \\ -500 \le x_2 \le 500. \end{gathered}$$

The global optimum has an objective value of $f_{\min} = -731.707$, located at $\mathbf{x} = (-394.15, -433.48)$. Only the third constraint is active at this point.

The example demonstrates how to initialize and set the options arrays using nag_glopt_opt_set (e05zkc), how to query options using nag_glopt_opt_get (e05zlc), and finally how to search for the global optimum using nag_glopt_nlp_pso (e05sbc). The problem is solved twice, first using nag_glopt_nlp_pso (e05sbc) alone, and secondly by coupling nag_glopt_nlp_pso (e05sbc) with nag_opt_nlp (e04ucc) as a dedicated local minimizer. In both cases the default option **Repeatability** $=$ ON is used to produce repeatable solutions.

10.1 Program Text

```
/* nag_glopt_nlp_pso (e05sbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 25, 2014.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage05.h>
#include <nagx02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL objfun_schwefel(Integer *mode, Integer ndim,
                                     const double x[], double *objf,
                                     double vecout[], Integer nstate,
                                     Nag_Comm *comm);
static void NAG_CALL confun(Integer *mode, Integer ncon, Integer ndim,
                            Integer tdcj, const Integer needc[],
                            const double x[], double c[], double cjac[],
                            Integer nstate, Nag_Comm *comm);
static void NAG_CALL monmod(Integer ndim, Integer ncon, Integer npar,
                            double x[], const double xb[], double fb,
                            const double cb[], const double xbest[],
                            const double fbest[], const double cbest[],
                            const Integer itt[], Nag_Comm *comm,
                            Integer *inform);
#ifdef __cplusplus
}
#endif

static Integer display_option(const char *optstr, const Integer iopts[],
                              const double opts[]);

static void display_result(Integer ndim, Integer ncon, const double xb[],
                           double fb, const double cb[], const Integer itt[],
                           Integer inform);

/* Global constants.*/
/* Set the behaviour of the monitoring function.*/
static const Integer detail_level = 0;
static const Integer report_freq = 100;
/* Known solution for a comparison.*/
static const double  f_target = -731.70709230672696;
static const double  c_scale[] = { 2490.0, 750000.0, 0.1 };
static const double  c_target[] = { 0.0, 0.0, 0.0 };
static const double  x_target[] = { -394.1470221120988, -433.48214189947606 };

int main(void)
{
  /* This example program demonstrates how to use
   * nag_glopt_nlp_pso (e05sbc) in standard execution, and with
   * nag_opt_nlp (e04ucc) as a coupled local minimizer.
   * The non-default option 'Repeatability = On' is used here, giving
   * repeatable results.
   */
  /* Scalars */
  Integer          ncon = 3, ndim = 2, npar = 20;
  Integer          exit_status = 0, lcvalue = 17;
  Integer          liopts = 100, lopts = 100;
  double           fb, rvalue;
  Integer          i, inform, ivalue;
  /* Arrays */
  static double ruser[3] = {-1.0, -1.0, -1.0};
  char             cvalue[17], optstr[81];
  double           opts[100], *bl = 0, *bu = 0, *cb = 0, *xb = 0;
  double           *cbest = 0, *fbest = 0, *xbest = 0;
  Integer          iopts[100], itt[7];
  /* Nag Types */
  Nag_VariableType optype;
  Nag_Comm         comm;
  NagError         fail;

  /* Print advisory information.*/
  printf("nag_glopt_nlp_pso (e05sbc) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  printf("Minimization of the Schwefel function.\n");
  printf("Subject to one linear and two nonlinear constraints.\n\n");

  /* Allocate memory for arrays.*/
  if (!(bl = NAG_ALLOC(ndim+ncon, double)) ||
      !(bu = NAG_ALLOC(ndim+ncon, double)) ||
      !(cb = NAG_ALLOC(ncon, double)) ||
      !(cbest = NAG_ALLOC(ncon*npar, double)) ||
      !(fbest = NAG_ALLOC(npar, double)) ||
      !(xb = NAG_ALLOC(ndim, double)) ||
      !(xbest = NAG_ALLOC(ndim*npar, double)))
```

```
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  for (i = 0; i < npar; i++)
    fbest[i] = 0.0;
  for (i = 0; i < npar*ndim; i++)
    xbest[i] = 0.0;
  for (i = 0; i < npar*ncon; i++)
    cbest[i] = 0.0;

  /* Set problem specific values.*/
  /* Set box bounds.*/
  for (i = 0; i < ndim; i++) {
    bl[i] = -500.0;
    bu[i] = 500.0;
  }

  /* Set constraint bounds.*/
  bl[ndim]   = -1.0e6;
  bl[ndim+1] = -1.0;
  bl[ndim+2] = -0.9;
  bu[ndim]   = 10.0;
  bu[ndim+1] = 5.0e5;
  bu[ndim+2] = 0.9;

  /* Initialize NagError structure.*/
  INIT_FAIL(fail);

  /* Initialize the option arrays for nag_glopt_nlp_pso (e05sbc)
   * using nag_glopt_opt_set (e05zkc).
   */
  nag_glopt_opt_set("Initialize = e05sbc", iopts, liopts, opts, lopts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_set (e05zkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Query some default option values.*/
  printf("  Default Option Queries:\n\n");
  exit_status = display_option("Constraint Norm", iopts, opts);
  if (exit_status) goto END;
  exit_status = display_option("Maximum Iterations Completed", iopts, opts);
  if (exit_status) goto END;
  exit_status = display_option("Distance Tolerance", iopts, opts);
  if (exit_status) goto END;

  /* ------------------------------------------------------------------*/
  printf("\n1. Solution without using coupled local minimizer.\n\n");

  /* Set various options to non-default values if required.*/
  nag_glopt_opt_set("Distance Tolerance = 1.0e-5", iopts, liopts, opts, lopts,
                    &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Constraint Tolerance = 1.0e-4", iopts, liopts, opts,
                      lopts, &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Constraint Norm = Euclidean", iopts, liopts, opts, lopts,
                      &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Repeatability = On", iopts, liopts, opts, lopts, &fail);
#ifdef _WIN32
  sprintf_s(optstr, _countof(optstr), "Target Objective Value = %32.16e",
            f_target);
#else
  sprintf(optstr, "Target Objective Value = %32.16e", f_target);
#endif
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Target Objective Tolerance = 1.0e-4", iopts, liopts,
                      opts, lopts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_set (e05zkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_glopt_nlp_pso (e05sbc).
   * Global optimization using particle swarm algorithm (PSO), comprehensive.
   */
  nag_glopt_nlp_pso(ndim, ncon, npar, xb, &fb, cb, bl, bu, xbest,
                    fbest, cbest, objfun_schwefel, confun,
                    monmod, iopts, opts, &comm, itt, &inform, &fail);
  /* It is essential to test fail.code on exit.*/
  switch (fail.code)
    {
    case NE_NOERROR:
    case NW_FAST_SOLUTION:
    case NW_SOLUTION_NOT_GUARANTEED:
      /* No errors, best found solution at xb returned in fb.*/
      display_result(ndim, ncon, xb, fb, cb, itt, inform);
      break;
    case NE_USER_STOP:
      /* Exit flag set in objfun, confun or monmod and returned in inform.*/
      display_result(ndim, ncon, xb, fb, cb, itt, inform);
      break;
    default: /* An error was detected.*/
      exit_status = 1;
      printf("Error from nag_glopt_nlp_pso (e05sbc)\n%s\n", fail.message);
      goto END;
    }

  /* ------------------------------------------------------------------*/

  printf("2. Solution using coupled local minimizer nag_opt_nlp (e04ucc).\n\n");

  /*  Set the local minimizer to be nag_opt_nlp (e04ucc) and set corresponding
   *  options.
   */
  nag_glopt_opt_set("Local Minimizer = e04ucc", iopts, liopts, opts, lopts,
                    &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Local Interior Major Iterations = 15", iopts, liopts,
                      opts, lopts, &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Local Interior Minor Iterations = 5", iopts, liopts,
                      opts, lopts, &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Local Exterior Major Iterations = 50", iopts, liopts,
                      opts, lopts, &fail);
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set("Local Exterior Minor Iterations = 15", iopts, liopts,
                      opts, lopts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_set (e05zkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Query the option Distance Tolerance*/
  nag_glopt_opt_get("Distance Tolerance", &ivalue, &rvalue, cvalue, lcvalue,
                    &optype, iopts, opts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_get (e05zlc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Adjust Distance Tolerance dependent upon its current value*/
  rvalue = rvalue*10.0;
#ifdef _WIN32
  sprintf_s(optstr, _countof(optstr), "Distance Tolerance = %32.16e", rvalue);
#else
  sprintf(optstr, "Distance Tolerance = %32.16e", rvalue);
#endif
  nag_glopt_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
  rvalue = 0.1*rvalue;
#ifdef _WIN32
  sprintf_s(optstr, _countof(optstr), "Local Interior Tolerance = %32.16e",
            rvalue);
#else
  sprintf(optstr, "Local Interior Tolerance = %32.16e", rvalue);
#endif
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
  rvalue = rvalue*1.0e-4;
#ifdef _WIN32
  sprintf_s(optstr, _countof(optstr), "Local Exterior Tolerance = %32.16e",
            rvalue);
#else
  sprintf(optstr, "Local Exterior Tolerance = %32.16e", rvalue);
#endif
  if (fail.code == NE_NOERROR)
    nag_glopt_opt_set(optstr, iopts, liopts, opts, lopts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_set (e05zkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_glopt_nlp_pso (e05sbc).
   * Global optimization using particle swarm algorithm (PSO), comprehensive.
   */
  nag_glopt_nlp_pso(ndim, ncon, npar, xb, &fb, cb, bl, bu, xbest,
                    fbest, cbest, objfun_schwefel, confun,
                    monmod, iopts, opts, &comm, itt, &inform, &fail);
  /* It is essential to test fail.code on exit.*/
  switch (fail.code)
    {
    case NE_NOERROR:
    case NW_FAST_SOLUTION:
    case NW_SOLUTION_NOT_GUARANTEED:
    case NW_NOT_FEASIBLE:
      /* nag_glopt_nlp_pso (e05sbc) encountered no errors during
       * operation, and will have returned the best solution found.
       */
      display_result(ndim, ncon, xb, fb, cb, itt, inform);
      break;
    case NE_USER_STOP:
      /* Exit flag set in objfun, confun or monmod and returned in inform.*/
      display_result(ndim, ncon, xb, fb, cb, itt, inform);
      break;
    default: /* An error was detected.*/
      exit_status = 1;
      printf("Error from nag_glopt_nlp_pso (e05sbc)\n%s\n", fail.message);
      goto END;
    }

 END:
  /* Clean up memory.*/
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(cb);
  NAG_FREE(cbest);
  NAG_FREE(fbest);
  NAG_FREE(xb);
  NAG_FREE(xbest);

  return exit_status;
}
```

```
static void NAG_CALL objfun_schwefel(Integer *mode, Integer ndim,
                                     const double x[], double *objf,
                                     double vecout[], Integer nstate,
                                     Nag_Comm *comm)
{
  /* Objective function routine returning the schwefel function and
   * its gradient.
   */
  Nag_Boolean evalobjf, evalobjg;
  Integer     i;
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback objfun_schwefel, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  /* Test nstate to indicate what stage of computation has been reached.*/
  switch (nstate)
    {
    case 2:
      /* objfun is called for the very first time. */
      break;
    case 1:
      /* objfun is called on entry to a NAG local minimizer. */
      break;
    default:
      /* This will be the normal value of nstate. */
      ;
    }
  /* Test mode to determine whether to calculate objf and/or objgrd.*/
  evalobjf = Nag_FALSE;
  evalobjg = Nag_FALSE;
  switch (*mode)
    {
    case 0:
    case 5:
      /* Only the value of the objective function is needed.*/
      evalobjf = Nag_TRUE;
      break;
    case 1:
    case 6:
      /* Only the values of the ndim gradients are required.*/
      evalobjg = Nag_TRUE;
      break;
    case 2:
    case 7:
      /* Both the objective function and the ndim gradients are required.*/
      evalobjf = Nag_TRUE;
      evalobjg = Nag_TRUE;
    }
  if (evalobjf)
    {
      /* Evaluate the objective function.*/
      *objf = 0.0;
      for (i = 0; i < ndim; i++)
        *objf += x[i]*sin(sqrt(fabs(x[i])));
    }
  if (evalobjg)
    {
      /* Calculate the gradient of the objective function, */
      /* and return the result in vecout.*/
      for (i = 0; i < ndim; i++)
        {
          vecout[i] = sqrt(fabs(x[i]));
          vecout[i] = sin(vecout[i]) + 0.5*vecout[i]*cos(vecout[i]);
        }
    }
}

static void NAG_CALL confun(Integer *mode, Integer ncon, Integer ndim,
                            Integer tdcj, const Integer needc[],
                           const double x[], double c[], double cjac[],
                           Integer nstate, Nag_Comm *comm)
{
  /* Supplies constraints.
   * cjac[(k-1)*tdcj + (i-1)] corresponds to dc[k]/dx[i]
   * for k=1,...,ncon and i=1,...,ndim.
   */
  Integer     k;
  Nag_Boolean evalc, evalcjac;

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback confun, first invocation.)\n");
      comm->user[1] = 0.0;
    }

  /* Test nstate to determine whether the local minimizer is being called
   * for the first time from a new start point.
   */
  if (nstate == 1) {
    /* Set any constant elements of the Jacobian matrix.*/
    cjac[0] = 3.0;
    cjac[1] = -2.0;
  }
  /* mode: are constraints, derivatives, or both are required? */
  evalc = (*mode == 0 || *mode == 2) ? Nag_TRUE : Nag_FALSE;
  evalcjac = (*mode == 1 || *mode == 2) ? Nag_TRUE : Nag_FALSE;

  for (k = 0; k < ncon; k++) {
    if (needc[k] <= 0)
      continue;

    if (evalc == Nag_TRUE) {
      /* Constraint values are required.
       * Only those for which needc is non-zero need be set.
       */
      switch (k)
        {
        case 0:
          c[k] = 3.0*x[0] - 2.0*x[1];
          break;
        case 1:
          c[k] = x[0]*x[0] - x[1]*x[1] + 3.0*x[0]*x[1];
          break;
        case 2:
          c[k] = cos(pow((x[0]/200.0), 2) + (x[1]/100.0));
          break;
        default:
          /* This constraint is not coded (there are only three).
           * Terminate.
           */
          *mode = -1;
          break;
        }
    }
    if (*mode < 0)
      break;
    if (evalcjac == Nag_TRUE) {
      /* Constraint derivatives (cjac) are required.*/
      switch (k)
        {
        case 0:
          /* Constant derivatives set when nstate=1 remain throughout
           * the local minimization.
           */
          break;
        case 1:
          /* If the constraint derivatives are known and are readily
           * calculated, populate cjac when required.
           */
          cjac[k*tdcj] = 2.0*x[0] + 3.0*x[1];
          cjac[k*tdcj+1] = -2.0*x[1] + 3.0*x[0];
          break;
        default:
          /* Any elements of cjac left unaltered will be approximated
           * using finite differences when required.
           */
          ;
        }
    }
  }
}

static void NAG_CALL monmod(Integer ndim, Integer ncon, Integer npar,
                            double x[], const double xb[], double fb,
                            const double cb[], const double xbest[],
                            const double fbest[], const double cbest[],
                            const Integer itt[], Nag_Comm *comm,
                            Integer *inform)
{
  Integer i, j;
#define X(J, I) x[(J-1)*ndim + (I-1)]
#define XBEST(J, I) xbest[(J-1)*ndim + (I-1)]
#define CBEST(J, I) cbest[(J-1)*ncon + (I-1)]
  if (comm->user[2] == -1.0)
    {
      printf("(User-supplied callback monmod, first invocation.)\n");
      comm->user[2] = 0.0;
    }
  if (detail_level)
    {
      /* Report on the first iteration, and every report_freq iterations.*/
      if (itt[0] == 1 || itt[0]%report_freq == 0)
        {
          printf("* Locations of particles\n");
          for (j = 1; j <= npar; j++)
            {
              printf("  * Particle %2"NAG_IFMT"\n", j);
              for (i = 1; i <= ndim; i++)
                printf("    %2"NAG_IFMT"  %13.5f\n", i, X(j, i));
            }
          printf("* Cognitive memory\n");
          for (j = 1; j <= npar; j++)
            {
              printf("  * Particle %2"NAG_IFMT"\n", j);
              printf("    * Best position\n");
              for (i = 1; i <= ndim; i++)
                printf("      %2"NAG_IFMT"  %13.5f\n", i, XBEST(j, i));
              printf("    * Function value at best position\n");
              printf("      %13.5f\n", fbest[j - 1]);
              printf("    * Best constraint violations\n");
              for (i = 1; i <= ncon; i++)
                printf("      %2"NAG_IFMT"  %13.5f\n", i, CBEST(j, i));
            }
          printf("* Current global optimum candidate\n");
          for (i = 1; i <= ndim; i++)
            printf("  %2"NAG_IFMT"  %13.5f\n", i, xb[i - 1]);
          printf("* Current global optimum value\n");
          printf("     %13.5f\n\n", fb);
          printf("* Constraint violations of candidate\n");
          for (i = 1; i <= ncon; i++)
            printf("  %2"NAG_IFMT"  %13.5f\n", i, cb[i - 1]);
        }
    }
  /* If required set *inform<0 to force exit.*/
  *inform = 0;
#undef CBEST
#undef XBEST
#undef X
}

static Integer display_option(const char *optstr, const Integer iopts[],
                              const double opts[])
{
  /* Subroutine to query optype and print the appropriate option values.*/

  /* Scalars */
  Integer          exit_status = 0, lcvalue = 17;
  double           rvalue = 0.0;
  Integer          ivalue = 0;
  /* Arrays */
  char             cvalue[17];
  /* Nag Types */
  Nag_VariableType optype = 0;
  NagError         fail;

  INIT_FAIL(fail);

  /* nag_glopt_opt_get (e05zlc).
   * Option getting routine for nag_glopt_nlp_pso (e05sbc).
   */
  nag_glopt_opt_get(optstr, &ivalue, &rvalue, cvalue, lcvalue, &optype, iopts,
                    opts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_glopt_opt_get (e05zlc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  switch (optype)
    {
    case Nag_Integer:
      printf("%-38s: %13"NAG_IFMT"\n", optstr, ivalue);
      break;
    case Nag_Real:
      printf("%-38s: %13.4f\n", optstr, rvalue);
      break;
    case Nag_Character:
      printf("%-38s: %13s\n", optstr, cvalue);
      break;
    case Nag_Integer_Additional:
      printf("%-38s: %13"NAG_IFMT" %16s\n", optstr, ivalue, cvalue);
      break;
    case Nag_Real_Additional:
      printf("%-38s: %13.4f %16s\n", optstr, rvalue, cvalue);
      break;
    default:;
    }
 END:
  return exit_status;
}

static void display_result(Integer ndim, Integer ncon, const double xb[],
                           double fb, const double cb[], const Integer itt[],
                           Integer inform)
{
  /* Display final results in comparison to known global optimum.*/
  Integer i;

  /* Display final counters.*/
  printf(" Algorithm Statistics\n");
  printf(" --------------------\n");
  printf("%-38s: %13"NAG_IFMT"\n", "Total complete iterations", itt[0]);
  printf("%-38s: %13"NAG_IFMT"\n", "Complete iterations since improvement",
         itt[1]);
  printf("%-38s: %13"NAG_IFMT"\n", "Total particles converged to xb", itt[2]);
  printf("%-38s: %13"NAG_IFMT"\n", "Total improvements to global optimum",
         itt[3]);
  printf("%-38s: %13"NAG_IFMT"\n", "Total function evaluations", itt[4]);
  printf("%-38s: %13"NAG_IFMT"\n", "Total particles re-initialized",
         itt[5]);
  printf("%-38s: %13"NAG_IFMT"\n\n", "Total constraints violated",
         itt[6]);
  /* Display why finalization occurred.*/
  switch (inform)
    {
    case 1:
      printf("Solution Status : Target value achieved\n");
      break;
    case 2:
      printf("Solution Status : Minimum swarm standard deviation obtained\n");
      break;
    case 3:
      printf("Solution Status : Sufficient number of particles converged\n");
      break;
    case 4:
      printf("Solution Status : Maximum static iterations attained\n");
      break;
    case 5:
      printf("Solution Status : Maximum complete iterations attained\n");
      break;
    case 6:
      printf("Solution Status : Maximum function evaluations exceeded\n");
      break;
    case 7:
      printf("Solution Status : Feasible point located\n");
      break;
    default:
      if (inform < 0) {
        printf("Solution Status : User termination, inform = %16"NAG_IFMT"\n",
               inform);
        return;
      }
      printf("Solution Status : Termination, an error has been detected\n");
      break;
    }
  /* Display final objective value and location.*/
  printf("  Known constrained objective optimum : %13.3f\n", f_target);
  printf("  Achieved objective value            : %13.3f\n\n", fb);

  printf("  Comparison between the known optimum and the achieved solution.\n");
  printf("          x_target            xb\n");
  for (i = 0; i < ndim; i++)
    printf("  %2"NAG_IFMT"  %12.2f  %12.2f\n", i+1, x_target[i], xb[i]);

  printf("\n");

  if (ncon > 0) {
    printf("  Comparison between scaled constraint violations.\n");
    printf("          c_target            cb\n");
    for (i = 0; i < ncon; i++)
      printf("  %2"NAG_IFMT"  %12.5f  %12.5f\n", i+1, c_target[i]/c_scale[i],
             cb[i]/c_scale[i]);

    printf("\n");
  }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_glopt_nlp_pso (e05sbc) Example Program Results

Minimization of the Schwefel function.
Subject to one linear and two nonlinear constraints.

  Default Option Queries:

Constraint Norm                      :             L1
Maximum Iterations Completed         :           1000          DEFAULT
```

```
Distance Tolerance                     :        0.0001

1. Solution without using coupled local minimizer.

(User-supplied callback confun, first invocation.)
(User-supplied callback objfun_schwefel, first invocation.)
(User-supplied callback monmod, first invocation.)
 Algorithm Statistics
 --------------------
Total complete iterations              :           277
Complete iterations since improvement  :             1
Total particles converged to xb        :             0
Total improvements to global optimum   :           117
Total function evaluations             :          4222
Total particles re-initialized         :             0
Total constraints violated             :             0

Solution Status : Target value achieved
  Known constrained objective optimum :      -731.707
  Achieved objective value            :      -731.708

  Comparison between the known optimum and the achieved solution.
          x_target           xb
   1       -394.15      -394.17
   2       -433.48      -433.53

  Comparison between scaled constraint violations.
          c_target           cb
   1       0.00000      0.00000
   2       0.00000      0.00000
   3       0.00000      0.00002

2. Solution using coupled local minimizer nag_opt_nlp (e04ucc).

 Algorithm Statistics
 --------------------
Total complete iterations              :             4
Complete iterations since improvement  :             1
Total particles converged to xb        :             0
Total improvements to global optimum   :             7
Total function evaluations             :           155
Total particles re-initialized         :             0
Total constraints violated             :             0

Solution Status : Target value achieved
  Known constrained objective optimum :      -731.707
  Achieved objective value            :      -731.706

  Comparison between the known optimum and the achieved solution.
          x_target           xb
   1       -394.15      -394.15
   2       -433.48      -433.49

  Comparison between scaled constraint violations.
          c_target           cb
   1       0.00000      0.00000
   2       0.00000      0.00000
   3       0.00000      0.00000
```

11 Algorithmic Details

The following pseudo-code describes the algorithm used with the repulsion mechanism.

```
INITIALIZE  for $j = 1,\ npar$
                $\mathbf{x}_j = \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$
                $\hat{\mathbf{x}}_j = \begin{cases} \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}}) & \textbf{Start} = \text{COLD} \\ \hat{\mathbf{x}}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$
                $\mathbf{v}_j = \mathbf{R} \in U(-\mathbf{V}_{\max}, \mathbf{V}_{\max})$
                $\hat{f}_j = \begin{cases} F(\hat{\mathbf{x}}_j) & \textbf{Start} = \text{COLD} \\ \hat{f}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$
                $\hat{\mathbf{e}}_j = \begin{cases} \mathbf{e}(\hat{\mathbf{x}}_j) & \textbf{Start} = \text{COLD} \\ \hat{\mathbf{e}}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$
                $w_j = \begin{cases} W_{\max} & \textbf{Weight Initialize} = \text{MAXIMUM} \\ W_{\text{ini}} & \textbf{Weight Initialize} = \text{INITIAL} \\ R \in U(W_{\min}, W_{\max}) & \textbf{Weight Initialize} = \text{RANDOMIZED} \end{cases}$
            end for
            $\tilde{\mathbf{x}} = \frac{1}{2}(\boldsymbol{\ell}_{\text{box}} + \mathbf{u}_{\text{box}})$
            $\tilde{f} = F(\tilde{\mathbf{x}})$
            $\tilde{\mathbf{e}} = \mathbf{e}(\tilde{\mathbf{x}})$
            $I_c = I_s = 0$

SWARM       while (not finalized),
                $I_c = I_c + 1$
                for $j = 1, npar$
                    $\mathbf{x}_j = \text{BOUNDARY}(\mathbf{x}_j, \boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$
                    $f_j = F(\mathbf{x}_j)$
                    $\mathbf{e}_j = \mathbf{e}(\mathbf{x}_j)$
                    if $\left(f_j/f_{\text{scale}} + \phi(w_j)\|\mathbf{e}_j\| < \hat{f}_j/f_{\text{scale}} + \phi(w_j)\|\hat{\mathbf{e}}_j\|\right)$
                        $\hat{f}_j = f_j,\ \hat{\mathbf{x}}_j = \mathbf{x}_j$
                    if $((\|\mathbf{e}_j\| < \|\tilde{\mathbf{e}}\|)$ or $(\|\mathbf{e}_j\| \approx \|\tilde{\mathbf{e}}\|$ and $f_j < \tilde{f}))$
                        $\tilde{f} = f_j,\ \tilde{\mathbf{x}} = \mathbf{x}_j$
                end for
                if $(\text{new}(\tilde{f}))$
                    LOCMIN $(\tilde{\mathbf{x}}, \tilde{f}, \tilde{\mathbf{e}}, O_i)$, $I_s = 0$
                [see note on repulsion below for code insertion]
                else
                    $I_s = I_s + 1$
                for $j = 1, npar$
                    $\mathbf{v}_j = w_j\mathbf{v}_j + C_s\mathbf{D}_1(\hat{\mathbf{x}}_j - \mathbf{x}_j) + C_g\mathbf{D}_2(\tilde{\mathbf{x}} - \mathbf{x}_j)$
                    $\mathbf{x}_j = \mathbf{x}_j + \mathbf{v}_j$
                    if $(\|\mathbf{x}_j - \tilde{\mathbf{x}}\| < dtol)$
                        reset $\mathbf{x}_j,\ \mathbf{v}_j,\ w_j;\ \hat{\mathbf{x}}_j = \mathbf{x}_j$
                    else
                        update $(w_j)$
                end for
                if (target achieved or termination criterion satisfied)
                    finalized = true
                **monmod**$(\mathbf{x}_j)$
            end
            LOCMIN $(\tilde{\mathbf{x}}, \tilde{f}, \tilde{\mathbf{e}}, O_e)$
```

The definition of terms used in the above pseudo-code are as follows.

npar	the number of particles, **npar**
$\boldsymbol{\ell}_{\text{box}}$	array of **ndim** lower box bounds
$\mathbf{u}_{\text{box}}$	array of **ndim** upper box bounds

$\mathbf{x}_j$	position of particle j
$\hat{\mathbf{x}}_j$	best position found by particle j
$\tilde{\mathbf{x}}$	best position found by any particle
f_j	$F(\mathbf{x}_j)$
$\hat{f}_j$	$F(\hat{\mathbf{x}}_j)$, best value found by particle j
$\tilde{f}$	$F(\tilde{\mathbf{x}})$, best value found by any particle
$e_k(\mathbf{x})$	kth (scaled) constraint violation at $\mathbf{x}$, evaluated as $\min(c_k(\mathbf{x}) - l_{\mathbf{ndim}+k}, 0.0) + \max(c_k(\mathbf{x}) - u_{\mathbf{ndim}+k}, 0.0)$; this may be scaled by the maximum kth constraint found thus far
$\mathbf{e}(\mathbf{x})$	the array of **ncon** constraint violations, $e_k(\mathbf{x})$, for $k = 1, 2, \ldots, \mathbf{ncon}$, at a point $\mathbf{x}$
$\mathbf{e}_j$	$\mathbf{e}(\mathbf{x}_j)$, the array of constraint violations evaluated at $\mathbf{x}_j$
$\hat{\mathbf{e}}_j$	$\mathbf{e}(\hat{\mathbf{x}}_j)$, the array of constraint violations evaluated at $\hat{\mathbf{x}}_j$
$\tilde{\mathbf{e}}$	$\mathbf{e}(\tilde{\mathbf{x}})$, the array of constraint violations evaluated at $\tilde{\mathbf{x}}$
$\mathbf{v}_j$	velocity of particle j
w_j	weight on $\mathbf{v}_j$ for velocity update, decreasing according to **Weight Decrease**
$\mathbf{V}_{\max}$	maximum absolute velocity, dependent upon **Maximum Variable Velocity**
I_c	swarm iteration counter
I_s	iterations since $\tilde{\mathbf{x}}$ was updated
f_{scale}	objective function scaling defined by the options **Constraint Scaling**, **Objective Scaling** and **Objective Scale**.
$\mathbf{D}_1$,$\mathbf{D}_2$	diagonal matrices with random elements in range $(0, 1)$
C_s	the cognitive advance coefficient which weights velocity towards $\hat{\mathbf{x}}_j$, adjusted using **Advance Cognitive**
C_g	the global advance coefficient which weights velocity towards $\tilde{\mathbf{x}}$, adjusted using **Advance Global**
$dtol$	the **Distance Tolerance** for resetting a converged particle
$\mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$	an array of random numbers whose i-th element is drawn from a uniform distribution in the range $(\boldsymbol{\ell}_{\text{box}\,i}, \mathbf{u}_{\text{box}\,i})$, for $i = 1, 2, \ldots, \mathbf{ndim}$
O_i	local optimizer interior options
O_e	local optimizer exterior options
$\phi(w_j)$	a function of w_j designed to increasingly weight towards minimizing constraint violations as w_j decreases
$\text{LOCMIN}(\mathbf{x}, f, \mathbf{e}, O)$	apply local optimizer using the set of options O using the solution $(\mathbf{x}, f, \mathbf{e})$ as the starting point, if used (not default)
monmod	monitor progress and possibly modify $\mathbf{x}_j$
BOUNDARY	apply required behaviour for $\mathbf{x}_j$ outside bounding box, (see **Boundary**)
new $(\tilde{f})$	true if $\tilde{\mathbf{x}}$, $\tilde{\mathbf{c}}$, $\tilde{f}$ were updated at this iteration

Additionally a repulsion phase can be introduced by changing from the default values of options **Repulsion Finalize** (r_f), **Repulsion Initialize** (r_i) and **Repulsion Particles** (r_p). If the number of static particles is denoted n_s then the following can be inserted after the new($\tilde{f}$) check in the pseudo-code above.

```
else   if   $(n_s \geq r_p$ and $r_i \leq I_s \leq r_i + r_f)$
            LOCMIN $(\tilde{\mathbf{x}}, \tilde{f}, \tilde{\mathbf{e}}, O_i)$
            use $-C_g$ instead of $C_g$ in velocity updates
       if   $(I_s = r_i + r_f)$
            $I_s = 0$
```

12 Optional Arguments

This section can be skipped if you wish to use the default values for all optional arguments, otherwise, the following is a list of the optional arguments available and a full description of each optional argument is provided in Section 12.1.

Advance Cognitive

Advance Global

Boundary

Constraint Norm

Constraint Scale Maximum

Constraint Scaling

Constraint Superiority

Constraint Tolerance

Constraint Warning

Distance Scaling

Distance Tolerance

Function Precision

Local Boundary Restriction

Local Exterior Iterations

Local Exterior Major Iterations

Local Exterior Minor Iterations

Local Exterior Tolerance

Local Interior Iterations

Local Interior Major Iterations

Local Interior Minor Iterations

Local Interior Tolerance

Local Minimizer

Maximum Function Evaluations

Maximum Iterations Completed

Maximum Iterations Static

Maximum Iterations Static Particles

Maximum Particles Converged

Maximum Particles Reset

Maximum Variable Velocity

Objective Scale

Objective Scaling

Optimize

Repeatability

Repulsion Finalize

Repulsion Initialize

Repulsion Particles
Start
Swarm Standard Deviation
Target Objective
Target Objective Safeguard
Target Objective Tolerance
Target Objective Value
Target Warning
Verify Gradients
Weight Decrease
Weight Initial
Weight Initialize
Weight Maximum
Weight Minimum
Weight Reset
Weight Value

12.1 Description of the Optional Arguments

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

The summary line contains:

the keywords;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value, where the symbol ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)), and $Imax$ represents the largest representable integer value (see nag_max_integer (X02BBC)).

All options accept the value 'DEFAULT' in order to return single options to their default states.

Keywords and character values are case insensitive, however they must be separated by at least one space.

For nag_glopt_nlp_pso (e05sbc) the maximum length of the argument **cvalue** used by nag_glopt_opt_get (e05zlc) is 15.

Advance Cognitive r Default $= 2.0$

The cognitive advance coefficient, C_s. When larger than the global advance coefficient, this will cause particles to be attracted toward their previous best positions. Setting $r = 0.0$ will cause nag_glopt_nlp_pso (e05sbc) to act predominantly as a local optimizer. Setting $r > 2.0$ may cause the swarm to diverge, and is generally inadvisable. At least one of the global and cognitive coefficients must be nonzero.

Advance Global r Default $= 2.0$

The global advance coefficient, C_g. When larger than the cognitive coefficient this will encourage convergence toward the best solution yet found. Values $r \in (0, 1)$ will inhibit particles overshooting the optimum. Values $r \in [1, 2)$ cause particles to fly over the optimum some of the time. Larger values can prohibit convergence. Setting $r = 0.0$ will remove any attraction to the current optimum, effectively generating a Monte–Carlo multi-start optimization algorithm. At least one of the global and cognitive coefficients must be nonzero.

Boundary a Default $=$ FLOATING

Determines the behaviour if particles leave the domain described by the box bounds. This only affects the general PSO algorithm, and will not pass down to any NAG local minimizers chosen.

This option is only effective in those dimensions for which $\mathbf{bl}[i-1] \neq \mathbf{bu}[i-1]$, $i = 1, 2, \ldots, \mathbf{ndim}$.

IGNORE
: The box bounds are ignored. The objective function is still evaluated at the new particle position.

RESET
: The particle is re-initialized inside the domain. $\hat{\mathbf{x}}_j$, $\hat{f}_j$ and $\hat{\mathbf{e}}_j$ are not affected.

FLOATING
: The particle position remains the same, however the objective function will not be evaluated at the next iteration. The particle will probably be advected back into the domain at the next advance due to attraction by the cognitive and global memory.

HYPERSPHERICAL
: The box bounds are wrapped around an $ndim$-dimensional hypersphere. As such a particle leaving through a lower bound will immediately re-enter through the corresponding upper bound and vice versa. The standard distance between particles is also modified accordingly.

FIXED
: The particle rests on the boundary, with the corresponding dimensional velocity set to 0.0.

Constraint Norm a Default $=$ L1

Determines with respect to which norm the constraint residuals should be constructed. These are automatically scaled with respect to **ncon** as stated. For the set of (scaled) violations $\mathbf{e}$, these may be,

L1
: The L^1 norm will be used, $\|\mathbf{e}\|_1 = \frac{1}{\mathbf{ncon}}\sum_1^{\mathbf{ncon}}|e_k|$

L2
: The L^2 norm will be used, $\|\mathbf{e}\|_2 = \frac{1}{\mathbf{ncon}}\sqrt{\sum_1^{\mathbf{ncon}} e_k^2}$

L2SQ
: The square of the L^2 norm will be used, $\|\mathbf{e}\|_{2^2} = \frac{1}{\mathbf{ncon}}\sum_1^{\mathbf{ncon}} e_k^2$

LMAX
: The L^∞ norm will be used, $\|\mathbf{e}\|_\infty = \max_{0<k\leq\mathbf{ncon}}(|e_k|)$

Constraint Scale Maximum r Default $=$ 1.0e6

Internally, each constraint violation is scaled with respect to the maximum violation yet achieved for that constraint. This option acts as a ceiling for this scale.

Constraint: $r > 1.0$.

Constraint Scaling a Default $=$ INITIAL

Determines whether to scale the constraints and objective function when constructing the penalty function.

OFF
: Neither the constraint violations nor the objective will be scaled automatically. This should only be used if the constraints and objective are similarly scaled everywhere throughout the domain.

INITIAL

The maximum of the initial cognitive memories, $\hat{f}_j$ and $\hat{\mathbf{e}}_j$, will be used to scale the objective function and constraint violations respectively.

ADAPTIVE

Initially, the maximum of the initial cognitive memories, $\hat{f}_j$ and $\hat{\mathbf{e}}_j$, will be used to scale the objective function and constraint violations respectively. If a significant change is detected in the behaviour of the constraints or the objective, these will be rescaled with respect to the current state of the cognitive memory.

Constraint Superiority r Default $= 0.01$

The minimum scaled improvement in the constraint violation for a location to be immediately superior to that in memory, regardless of the objective value.

Constraint: $r > 0.0$.

Constraint Tolerance r Default $= 10^{-4}$

The maximum scaled violation of the constraints for which a sample particle is considered comparable to the current global optimum. Should this not be exceeded, then the current global optimum will be updated if the value of the objective function of the sample particle is superior.

Constraint Warning a Default $=$ ON

Activates or deactivates the error exit associated with the inability to completely satisfy all constraints, **fail.code** $=$ NW_NOT_FEASIBLE. It is advisable to deactivate this option if the exit **fail.code** $=$ NW_NOT_FEASIBLE is preferred in such cases.

OFF

fail.code $=$ NW_NOT_FEASIBLE will not be returned.

ON

fail.code $=$ NW_NOT_FEASIBLE will be returned if any constraints are sufficiently violated at the end of the simulation.

Distance Scaling a Default $=$ ON

Determines whether distances should be scaled by box widths.

ON

When a distance is calculated between $\mathbf{x}$ and $\mathbf{y}$, a scaled L^2 norm is used.

$$L^2(\mathbf{x},\mathbf{y}) = \left(\sum_{\{i|\mathbf{u}_i\neq\boldsymbol{\ell}_i, i\leq ndim\}} \left(\frac{x_i - y_i}{\mathbf{u}_i - \boldsymbol{\ell}_i}\right)^2\right)^{\frac{1}{2}}.$$

OFF

Distances are calculated as the standard L^2 norm without any rescaling.

$$L^2(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{ndim} (x_i - y_i)^2\right)^{\frac{1}{2}}.$$

Distance Tolerance r Default $= 10^{-4}$

This is the distance, $dtol$ between particles and the global optimum which must be reached for the particle to be considered converged, i.e., that any subsequent movement of such a particle cannot significantly alter the global optimum. Once achieved the particle is reset into the box bounds to continue searching.

Constraint: $r > 0.0$.

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

The argument defines ϵ_r, which is intended to be a measure of the accuracy with which the problem function $F(\mathbf{x})$ can be computed. If $r < \epsilon$ or $r \geq 1$, the default value is used.

The value of ϵ_r should reflect the relative precision of $1 + |F(\mathbf{x})|$; i.e., ϵ_r acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(\mathbf{x})$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-6}. In contrast, if $F(\mathbf{x})$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-10}. The choice of ϵ_r can be quite complicated for badly scaled problems; see Chapter 8 of Gill *et al.* (1981) for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_r should be large enough so that no attempt will be made to distinguish between function values that differ by less than the error inherent in the calculation.

Local Boundary Restriction r Default $= 0.5$

Contracts the box boundaries used by a box constrained local minimizer to, $[\beta_l, \beta_u]$, containing the start point x, where

$$\begin{aligned} \partial_i &= r \times (\mathbf{u}_i - \boldsymbol{\ell}_i) \\ \beta_l^i &= \max\left(\boldsymbol{\ell}_i, x_i - \tfrac{\partial_i}{2}\right) \\ \beta_u^i &= \min\left(\mathbf{u}_i, x_i + \tfrac{\partial_i}{2}\right), \quad i = 1, \ldots, \mathbf{ndim}. \end{aligned}$$

Smaller values of r thereby restrict the size of the domain exposed to the local minimizer, possibly reducing the amount of work done by the local minimizer.

Constraint: $0.0 \leq r \leq 1.0$.

Local Interior Iterations i_1
Local Interior Major Iterations i_1
Local Exterior Iterations i_2
Local Exterior Major Iterations i_2

The maximum number of iterations or function evaluations the chosen local minimizer will perform inside (outside) the main loop if applicable. For the NAG minimizers these correspond to:

Minimizer	**Argument/option**	**Default Interior**	**Default Exterior**
nag_opt_simplex_easy (e04cbc)	**maxcal**	$\mathbf{ndim} + 10$	$2 \times \mathbf{ndim} + 15$
nag_opt_conj_grad (e04dgc)	**Iteration Limit**	$\max(30, 3 \times \mathbf{ndim})$	$\max(50, 5 \times \mathbf{ndim})$
nag_opt_nlp (e04ucc)	**Major Iteration Limit**	$\max(10, 2 \times \mathbf{ndim})$	$\max(30, 3 \times \mathbf{ndim})$

Unless set, these are functions of the arguments passed to nag_glopt_nlp_pso (e05sbc).

Setting $i = 0$ will disable the local minimizer in the corresponding algorithmic region. For example, setting **Local Interior Iterations** $= 0$ and **Local Exterior Iterations** $= 30$ will cause the algorithm to perform no local minimizations inside the main loop of the algorithm, and a local minimization with upto 30 iterations after the main loop has been exited.

Constraint: $i_1 \geq 0$, $i_2 \geq 0$.

Local Interior Tolerance r_1 Default $= 10^{-4}$
Local Exterior Tolerance r_2 Default $= 10^{-4}$

This is the tolerance provided to a local minimizer in the interior (exterior) of the main loop of the algorithm.

Constraint: $r_1 > 0.0$, $r_2 > 0.0$.

Local Interior Minor Iterations i_1
Local Exterior Minor Iterations i_2

Where applicable, the secondary number of iterations the chosen local minimizer will use inside (outside) the main loop. Currently the relevant default values are:

Minimizer	**Argument/option**	**Default Interior**	**Default Exterior**
nag_opt_nlp (e04ucc)	**Minor Iteration Limit**	$\max(10, 2 \times \textbf{ndim})$	$\max(30, 3 \times \textbf{ndim})$

Constraint: $i_1 \geq 0$, $i_2 \geq 0$.

Local Minimizer a Default $=$ OFF

Allows for a choice of Chapter e04 functions to be used as a coupled, dedicated local minimizer.

OFF
: No local minimization will be performed in either the INTERIOR or EXTERIOR sections of the algorithm.

e04cbc
: Use nag_opt_simplex_easy (e04cbc) as the local minimizer. This does not require the calculation of derivatives.

On a call to **objfun** during a local minimization, **mode** $= 5$.

e04dgc
: Use nag_opt_conj_grad (e04dgc) as the local minimizer.

Accurate derivatives must be provided, and will not be approximated internally. Additionally, each call to **objfun** during a local minimization will require either the objective to be evaluated alone, or both the objective and its gradient to be evaluated. Hence on a call to **objfun**, **mode** $= 5$ or 7.

e04ucc
: Use nag_opt_nlp (e04ucc) as the local minimizer. This operates such that any derivatives of either the objective function or the constraint Jacobian, which you cannot supply, will be approximated internally using finite differences.

Either, the objective, objective gradient, or both may be requested during a local minimization, and as such on a call to **objfun**, **mode** $= 1$, 2 or 5.

The box bounds forwarded to this function from nag_glopt_nlp_pso (e05sbc) will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to nag_glopt_nlp_pso (e05sbc).

Maximum Function Evaluations i Default $= Imax$

The maximum number of evaluations of the objective function. When reached this will return **fail.code** $=$ NW_SOLUTION_NOT_GUARANTEED and **inform** $= 6$.

Constraint: $i > 0$.

Maximum Iterations Completed i Default $= 1000 \times$ **ndim**

The maximum number of complete iterations that may be performed. Once exceeded nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NW_SOLUTION_NOT_GUARANTEED and **inform** $= 5$.

Unless set, this adapts to the parameters passed to nag_glopt_nlp_pso (e05sbc).

Constraint: $i \geq 1$.

Maximum Iterations Static i Default $= 100$

The maximum number of iterations without any improvement to the current global optimum. If exceeded nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NW_SOLUTION_NOT_GUARANTEED and **inform** $= 4$. This exit will be hindered by setting **Maximum Iterations Static Particles** to larger values.

Constraint: $i \geq 1$.

Maximum Iterations Static Particles i Default $= 0$

The minimum number of particles that must have converged to the current optimum before the function may exit due to **Maximum Iterations Static** with **fail.code** = NW_SOLUTION_NOT_GUARANTEED and **inform** $= 4$.

Constraint: $i \geq 0$.

Maximum Particles Converged i Default $= \mathit{Imax}$

The maximum number of particles that may converge to the current optimum. When achieved, nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** = NW_SOLUTION_NOT_GUARANTEED and **inform** $= 3$. This exit will be hindered by setting '**Repulsion**' options, as these cause the swarm to re-expand.

Constraint: $i > 0$.

Maximum Particles Reset i Default $= \mathit{Imax}$

The maximum number of particles that may be reset after converging to the current optimum. Once achieved no further particles will be reset, and any particles within **Distance Tolerance** of the global optimum will continue to evolve as normal.

Constraint: $i > 0$.

Maximum Variable Velocity r Default $= 0.25$

Along any dimension j, the absolute velocity is bounded above by $|\mathbf{v}_j| \leq r \times (\mathbf{u}_j - \boldsymbol{\ell}_j) = \mathbf{V}_{\max}$. Very low values will greatly increase convergence time. There is no upper limit, although larger values will allow more particles to be advected out of the box bounds, and values greater than 4.0 may cause significant and potentially unrecoverable swarm divergence.

Constraint: $r > 0.0$.

Objective Scale r Default $= 1.0$

The initial scale for the objective function. This will remain fixed if **Objective Scaling** = USER is selected.

Objective Scaling a Default = MAXIMUM

The method of (re)scaling applied to the objective function when the function detects a significant difference between the scale and the global and cognitive memory ($\tilde{f}$ and $\hat{f}_j$). This only has an effect when **ncon** > 0 and **Constraint Scaling** is active.

MAXIMUM
: The objective is rescaled with respect to the maximum absolute value of the objective in the cognitive and global memory.

MEAN
: The objective is rescaled with respect to the mean absolute value of the objective in the cognitive and global memory.

USER
: The scale remains fixed at the value set using **Objective Scale**.

Optimize a Default = MINIMIZE

Determines whether to maximize or minimize the objective function, or ignore the objective and search for a constrained point.

MINIMIZE
: The objective function will be minimized.

MAXIMIZE

The objective function will be maximized. This is accomplished by minimizing the negative of the objective.

CONSTRAINTS

The objective function will be ignored, and the algorithm will attempt to find a feasible point given the provided constraints. The objective function will be evaluated at the best point found with regards to constraint violations, and the final positions returned in **xbest**. The objective will be calculated at the best point found in terms of constraints only. Should a constrained point be found, nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NE_NOERROR and **inform** $= 6$.

Constraint: if **Optimize** $=$ CONSTRAINTS, **ncon** > 0 is required.

Repeatability a Default $=$ OFF

Allows for the same random number generator seed to be used for every call to nag_glopt_nlp_pso (e05sbc). **Repeatability** $=$ OFF is recommended in general.

OFF

The internal generation of random numbers will be nonrepeatable.

ON

The same seed will be used.

Repulsion Finalize i Default $= Imax$

The number of iterations performed in a repulsive phase before re-contraction. This allows a re-diversified swarm to contract back toward the current optimum, allowing for a finer search of the near optimum space.

Constraint: $i \geq 2$.

Repulsion Initialize i Default $= Imax$

The number of iterations without any improvement to the global optimum before the algorithm begins a repulsive phase. This phase allows the particle swarm to re-expand away from the current optimum, allowing more of the domain to be investigated. The repulsive phase is automatically ended if a superior optimum is found.

Constraint: $i \geq 2$.

Repulsion Particles i Default $= 0$

The number of particles required to have converged to the current optimum before any repulsive phase may be initialized. This will prevent repulsion before a satisfactory search of the near optimum area has been performed, which may happen for large dimensional problems.

Constraint: $i \geq 0$.

Start a Default $=$ COLD

Used to affect the initialization of the function.

COLD

The random number generators and all initialization data will be generated internally. The variables **xbest**, **fbest** and **cbest** need not be set.

WARM

You must supply the initial best location, function and constraint violation values **xbest**, **fbest** and **cbest**. This option is recommended if you already have a data set you wish to improve upon.

Swarm Standard Deviation r Default $= 0.1$

The target standard deviation of the particle distances from the current optimum. Once the standard deviation is below this level, nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NW_SOLUTION_NOT_GUARANTEED and **inform** $= 2$. This criterion will be penalized by the use of ‘**Repulsion**’ options, as these cause the swarm to re-expand, increasing the standard deviation of the particle distances from the best point.

Constraint: $r \geq 0.0$.

Target Objective a Default $=$ OFF
Target Objective Value r Default $= 0.0$

Activate or deactivate the use of a target value as a finalization criterion. If active, then once the supplied target value for the objective function is found (beyond the first iteration if **Target Warning** is active) nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NE_NOERROR and **inform** $= 1$. Other than checking for feasibility only (**Optimize** $=$ CONSTRAINTS), this is the only finalization criterion that guarantees that the algorithm has been successful. If the target value was achieved at the initialization phase or first iteration and **Target Warning** is active, nag_glopt_nlp_pso (e05sbc) will exit with **fail.code** $=$ NW_FAST_SOLUTION. This option may take any real value r, or the character ON/OFF as well as DEFAULT. If this option is queried using nag_glopt_opt_get (e05zlc), the current value of r will be returned in **rvalue**, and **cvalue** will indicate whether this option is ON or OFF. The behaviour of the option is as follows:

r
: Once a point is found with an objective value within the **Target Objective Tolerance** of r, nag_glopt_nlp_pso (e05sbc) will exit successfully with **fail.code** $=$ NE_NOERROR and **inform** $= 1$.

OFF
: The current value of r will remain stored, however it will not be used as a finalization criterion.

ON
: The current value of r stored will be used as a finalization criterion.

DEFAULT
: The stored value of r will be reset to its default value (0.0), and this finalization criterion will be deactivated.

Target Objective Safeguard r Default $= 10.0\epsilon$

If you have given a target objective value to reach in *objval* (the value of the optional argument **Target Objective Value**), *objsfg* sets your desired safeguarded termination tolerance, for when *objval* is close to zero.

Constraint: $objsfg \geq 2\epsilon$.

Target Objective Tolerance r Default $= 0.0$

The optional tolerance to a user-specified target value.

Constraint: $r \geq 0.0$.

Target Warning a Default $=$ OFF

Activates or deactivates the error exit associated with the target value being achieved before entry into the main loop of the algorithm, **fail.code** $=$ NW_FAST_SOLUTION.

OFF
: No error will be returned, and the function will exit normally.

ON
: An error will be returned if the target objective is reached prematurely, and the function will exit with **fail.code** $=$ NW_FAST_SOLUTION.

Verify Gradients a Default $=$ ON

Adjusts the level of gradient checking performed when gradients are required. Gradient checks are only performed on the first call to the chosen local minimizer if it requires gradients. There is no guarantee that the gradient check will be correct, as the finite differences used in the gradient check are themselves subject to inaccuracies.

OFF
: No gradient checking will be performed.

ON
: A cheap gradient check will be performed on both the gradients corresponding to the objective through **objfun** and those provided via the constraint Jacobian through **confun**.

OBJECTIVE
: A more expensive gradient check will be performed on the gradients corresponding to the objective **objfun**. The gradients of the constraints will not be checked.

CONSTRAINTS
: A more expensive check will be performed on the elements of **cjac** provided via **confun**. The objective gradient will not be checked.

FULL
: A more expensive check will be performed on both the gradient of the objective and the constraint Jacobian.

Weight Decrease a Default $=$ INTEREST

Determines how particle weights decrease.

OFF
: Weights do not decrease.

INTEREST
: Weights decrease through compound interest as $w_{IT+1} = w_{IT}(1 - W_{val})$, where W_{val} is the **Weight Value** and IT is the current number of iterations.

LINEAR
: Weights decrease linearly following $w_{IT+1} = w_{IT} - IT \times (W_{max} - W_{min})/IT_{max}$, where IT is the iteration number and IT_{max} is the maximum number of iterations as set by **Maximum Iterations Completed**.

Weight Initial r Default $= W_{max}$

The initial value of any particle's inertial weight, W_{ini}, or the minimum possible initial value if initial weights are randomized. When set, this will override **Weight Initialize** $=$ RANDOMIZED or MAXIMUM, and as such these must be set afterwards if so desired.

Constraint: $W_{min} \leq r \leq W_{max}$.

Weight Initialize a Default $=$ MAXIMUM

Determines how the initial weights are distributed.

INITIAL
: All weights are initialized at the initial weight, W_{ini}, if set. If **Weight Initial** has not been set, this will be the maximum weight, W_{max}.

MAXIMUM
: All weights are initialized at the maximum weight, W_{max}.

RANDOMIZED
: Weights are uniformly distributed in (W_{min}, W_{max}) or (W_{ini}, W_{max}) if **Weight Initial** has been set.

Weight Maximum r Default $= 1.0$

The maximum particle weight, W_{max}.

Constraint: $1.0 \geq r \geq W_{min}$ (If W_{ini} has been set then $1.0 \geq r \geq W_{ini}$.)

Weight Minimum r Default $= 0.1$

The minimum achievable weight of any particle, W_{min}. Once achieved, no further weight reduction is possible.

Constraint: $0.0 \leq r \leq W_{max}$ (If W_{ini} has been set then $0.0 \leq r \leq W_{ini}$.)

Weight Reset a Default $=$ MAXIMUM

Determines how particle weights are re-initialized.

INITIAL
: Weights are re-initialized at the initial weight if set. If **Weight Initial** has not been set, this will be the maximum weight.

MAXIMUM
: Weights are re-initialized at the maximum weight.

RANDOMIZED
: Weights are uniformly distributed in (W_{min}, W_{max}) or (W_{ini}, W_{max}) if **Weight Initial** has been set.

Weight Value r Default $= 0.01$

The constant W_{val} used with **Weight Decrease** $=$ INTEREST.

Constraint: $0.0 \leq r \leq \frac{1}{3}$.