# NAG Library Routine Document

# E05SBF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

**Note**: *this routine uses* **optional parameters** *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 parameters, 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 parameters*.

## 1 Purpose

E05SBF 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 routine if desired. E05SBF is essentially identical to E05SAF, with an expert interface and various additional parameters added; otherwise most parameters are identical. In particular, E05SAF does not handle general constraints.

## 2 Specification

```
SUBROUTINE E05SBF (NDIM, NCON, NPAR, XB, FB, CB, BL, BU, XBEST, FBEST,     &
                   CBEST, OBJFUN, CONFUN, MONMOD, IOPTS, OPTS, IUSER,      &
                   RUSER, ITT, INFORM, IFAIL)
INTEGER            NDIM, NCON, NPAR, IOPTS(*), IUSER(*), ITT(7),           &
                   INFORM, IFAIL
REAL (KIND=nag_wp) XB(NDIM), FB, CB(NCON), BL(NDIM+NCON),                  &
                   BU(NDIM+NCON), XBEST(NDIM,NPAR), FBEST(NPAR),           &
                   CBEST(NCON,NPAR), OPTS(*), RUSER(*)
EXTERNAL           OBJFUN, CONFUN, MONMOD
```

Before calling E05SBF, E05ZKF **must** be called with OPTSTR set to '`Initialize = e05sbf`'. Optional parameters may also be specified by calling E05ZKF before the call to E05SBF.

## 3 Description

E05SBF 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 routines, 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 routines directly.

For functions which are smooth and multi-modal, gradient dependent local minimization routines may be coupled with E05SBF.

For multi-modal functions for which derivatives cannot be provided, particularly functions with a significant level of noise in their evaluation, E05SBF should be used either alone, or coupled with E04CBF.

For heavily constrained problems, E05SBF should either be used alone, or coupled with E04UCF/ E04UCA provided the function and the constraints are sufficiently smooth.

The $ndim$ lower and upper box bounds on the variable **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.

E05SBF 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, E05SBF will exit with IFAIL $= 1$, 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.

E05SAF 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 Parameters

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

1: NDIM – INTEGER *Input*

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

*Constraint*: NDIM $\geq 1$.

2: NCON – INTEGER *Input*

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

*Constraint*: 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*: NPAR $= 10 \times$ NDIM.

*Constraint*: NPAR $\geq 5 \times$ **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 routine.

4: XB(NDIM) – REAL (KIND=nag_wp) array *Output*

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

5: FB – REAL (KIND=nag_wp) *Output*

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

6: CB(NCON) – REAL (KIND=nag_wp) array *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) – REAL (KIND=nag_wp) array *Input*

8: BU(NDIM + NCON) – REAL (KIND=nag_wp) array *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 BL$(i)$ = BU$(i)$ for any $i \in \{1, \ldots, \text{NDIM}\}$, variable $i$ will remain locked to BL$(i)$ 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*:

BL$(i) \leq$ BU$(i)$, for $i = 1, 2, \ldots, \text{NDIM} + \text{NCON}$;
BL$(i) \neq$ BU$(i)$ for at least one $i \in \{1, \ldots, \text{NDIM}\}$.

9: XBEST(NDIM, NPAR) – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the $i$th component of the best position of the $j$th particle, $\hat{x}_j(i)$, is stored in XBEST$(i, j)$.

*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) – REAL (KIND=nag_wp) array *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, $\hat{f}_j = F\left(\hat{\mathbf{x}}_j\right)$, corresponding to the locations returned in XBEST.

11: CBEST(NCON, NPAR) – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the $k$th constraint violation of the $j$th particle is stored in CBEST$(k, j)$.

*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 – SUBROUTINE, supplied by the user. *External Procedure*

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:

```
SUBROUTINE OBJFUN (MODE, NDIM, X, OBJF, VECOUT, NSTATE, IUSER,      &
                   RUSER)

INTEGER            MODE, NDIM, NSTATE, IUSER(*)
REAL (KIND=nag_wp) X(NDIM), OBJF, VECOUT(NDIM), RUSER(*)
```

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** = E04UCF 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** = E04UCF 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** = E04DGF or E04KZF only
*All* first derivatives *must* be evaluated and returned in VECOUT.

MODE = 7
: **Local Minimizer** = E04DGF or E04KZF 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 E05SBF will exit as soon as possible with IFAIL = 3 and INFORM = MODE.

2: NDIM – INTEGER *Input*

*On entry*: the number of dimensions.

3: X(NDIM) – REAL (KIND=nag_wp) array *Input*

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

4: OBJF – REAL (KIND=nag_wp) *Input/Output*

*On entry*: the value of OBJF passed to OBJFUN varies with the parameter 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 parameter 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) – REAL (KIND=nag_wp) array *Input/Output*

*On entry*: if **Local Minimizer** = E04UCF or E04UCA, the values of VECOUT are used internally to indicate whether a finite difference approximation is required. See E04UCF/E04UCA.

*On exit*: the required values of VECOUT returned to the calling routine 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 E04DGF/E04DGA and E04KZF).

6: NSTATE – INTEGER *Input*

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

NSTATE = 3
SMP users only. OBJFUN is called for the first time in a parallel region on a new thread other than the master thread. You may use this opportunity to set up any thread-dependent information in IUSER and RUSER.

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 routine. 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: IUSER($*$) – INTEGER array *User Workspace*
8: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

OBJFUN is called with the parameters IUSER and RUSER as supplied to E05SBF. You are free to use the arrays IUSER and RUSER to supply information to OBJFUN as an alternative to using COMMON global variables.

OBJFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E05SBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: CONFUN – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

CONFUN must calculate any constraints other than the box constraints. If no constraints are required, CONFUN may be the dummy constraint routine E05SZM. (E05SZM is included in the NAG Library). For information on how a NAG local minimizer will use CONFUN see the documentation for E04UCA.

The specification of CONFUN is:

```
SUBROUTINE CONFUN (MODE, NCON, NDIM, LDCJ, NEEDC, X, C, CJAC,        &
                   NSTATE, IUSER, RUSER)
INTEGER            MODE, NCON, NDIM, LDCJ, NEEDC(NCON), NSTATE,      &
                   IUSER(*)
REAL (KIND=nag_wp) X(NDIM), C(NCON), CJAC(LDCJ,NDIM), RUSER(*)
```

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 $k \in \{1,\ldots,\mathrm{NCON}\}$ such that $\mathrm{NEEDC}(k) > 0$:

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 $i = 1,2,\ldots,\mathrm{NDIM}$.

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 $i = 1,2,\ldots,\mathrm{NDIM}$.

*On exit*: may be set to a negative value if you wish to terminate the solution to the current problem. In this case E05SBF will terminate with IFAIL $= 3$ 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: LDCJ – INTEGER *Input*

*On entry*: the first dimension of the array CJAC as declared in the (sub)program from which E05SBF is called.

5: NEEDC(NCON) – INTEGER array *Input*

*On entry*: the indices of the elements of C and/or CJAC that must be evaluated by CONFUN. If NEEDC$(k) > 0$, the $k$th element of C, corresponding to the values of the $k$th constraint, and/or the available elements of the $k$th row of CJAC, corresponding to the derivatives of the $k$th constraint, must be evaluated at **x** (see parameter MODE).

6: X(NDIM) – REAL (KIND=nag_wp) array *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) – REAL (KIND=nag_wp) array *Output*

*On exit*: if NEEDC$(k) > 0$ and MODE $= 0$ or 2, C$(k)$ 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(LDCJ, NDIM) – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the derivative of the $k$th constraint with respect to the $i$th component, $\dfrac{\partial c_k}{\partial x_{\mathrm{i}}}$, is stored in CJAC$(k, i)$.

*On entry*: the elements of CJAC are set to special values which enable E05SBF to detect whether they are changed by CONFUN.

*On exit*: if NEEDC$(k) > 0$ and MODE $= 1$ or 2, the elements of CJAC corresponding to the $k$th row of the constraint jacobian should contain the available elements of the vector $\nabla c_k$ 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 $\dfrac{\partial c_k}{\partial x_i}$ is the partial derivative of the $k$th constraint with respect to the $i$th 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 routine. This allows for permanent global parameters 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 $= 3$
: SMP users only. OBJFUN is called for the first time in a parallel region on a new thread other than the master thread. You may use this opportunity to set up any thread-dependent information in IUSER and RUSER.

NSTATE $= 2$
CONFUN is called for the very first time. This parameter 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 routine. This parameter 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: IUSER$(*)$ – INTEGER array *User Workspace*
11: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

CONFUN is called with the parameters IUSER and RUSER as supplied to E05SBF. You are free to use the arrays IUSER and RUSER to supply information to CONFUN as an alternative to using COMMON global variables.

CONFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E05SBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

CONFUN should be tested separately before being used in conjunction with E05SBF.

14: MONMOD – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

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 the dummy monitoring routine E05SYM. (E05SYM is included in the NAG Library) .

The specification of MONMOD is:

```
SUBROUTINE MONMOD (NDIM, NCON, NPAR, X, XB, FB, CB, XBEST, FBEST,    &
                   CBEST, ITT, IUSER, RUSER, INFORM)

INTEGER            NDIM, NCON, NPAR, ITT(7), IUSER(*), INFORM
REAL (KIND=nag_wp) X(NDIM,NPAR), XB(NDIM), FB, CB(NCON),             &
                   XBEST(NDIM,NPAR), FBEST(NPAR),                    &
                   CBEST(NCON,NPAR), RUSER(*)
```

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$(\mathrm{NDIM}, \mathrm{NPAR})$ – REAL (KIND=nag_wp) array *Input/Output*

**Note**: the $i$th component of the $j$th particle, $x_j(i)$, is stored in $\mathrm{X}(i, j)$.

*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) – REAL (KIND=nag_wp) array *Input*

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

6: FB – REAL (KIND=nag_wp) *Input*

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

7: CB(NCON) – REAL (KIND=nag_wp) array *Input*

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

8: XBEST(NDIM, NPAR) – REAL (KIND=nag_wp) array *Input*

**Note**: the $i$th component of the position of the $j$th particle's cognitive memory, $\hat{x}_j(i)$, is stored in XBEST$(i, j)$.

*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) – REAL (KIND=nag_wp) array *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) – REAL (KIND=nag_wp) array *Input*

**Note**: the $k$th constraint violation of the $j$th particle's cognitive memory is stored in CBEST$(k, j)$.

*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) – INTEGER array *Input*

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

12: IUSER(∗) – INTEGER array *User Workspace*

13: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

MONMOD is called with the parameters IUSER and RUSER as supplied to E05SBF. You are free to use the arrays IUSER and RUSER to supply information to MONMOD as an alternative to using COMMON global variables.

14: INFORM – INTEGER *Input/Output*

*On entry*: INFORM = **thread_num**, where **thread_num** is the value returned by a call of the OpenMP function `OMP_GET_THREAD_NUM()`. If running in serial this will always be zero.

*On exit*: setting INFORM $< 0$ will cause near immediate exit from E05SBF. This value will be returned as INFORM with IFAIL $= 3$. You need not set INFORM unless you wish to force an exit.

MONMOD must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E05SBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: IOPTS(∗) – INTEGER array *Communication Array*

**Note**: the dimension 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 E05ZKF.

*On entry*: optional parameter array as generated and possibly modified by calls to E05ZKF. The contents of IOPTS **must not** be modified directly between calls to E05SBF, E05ZKF or E05ZLF.

16: OPTS(∗) – REAL (KIND=nag_wp) array *Communication Array*

**Note**: the dimension 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 E05ZKF.

*On entry*: optional parameter array as generated and possibly modified by calls to E05ZKF. The contents of OPTS **must not** be modified directly between calls to E05SBF, E05ZKF or E05ZLF.

17: IUSER(∗) – INTEGER array *User Workspace*

IUSER is not used by E05SBF, but is passed directly to OBJFUN, CONFUN and MONMOD and may be used to pass information to these routines as an alternative to using COMMON global variables.

With care, you may also write information back into IUSER. This might be useful, for example, should there be a need to preserve the state of a random number generator.

With SMP-enabled versions of E05SBF the array IUSER provided are classified as OpenMP shared memory. Use of IUSER has to take account of this in order to preserve thread safety whenever information is written back to either of these arrays.

18: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

RUSER is not used by E05SBF, but is passed directly to OBJFUN, CONFUN and MONMOD and may be used to pass information to these routines as an alternative to using COMMON global variables.

With care, you may also write information back into RUSER. This might be useful, for example, should there be a need to preserve the state of a random number generator.

With SMP-enabled versions of E05SBF the array RUSER provided are classified as OpenMP shared memory. Use of RUSER has to take account of this in order to preserve thread safety whenever information is written back to either of these arrays.

19: ITT(7) – INTEGER array *Output*

*On exit*: integer iteration counters for E05SBF.

ITT(1)
Number of complete iterations.

ITT(2)
Number of complete iterations without improvement to the current optimum.

ITT(3)
Number of particles converged to the current optimum.

ITT(4)
Number of improvements to the optimum.

ITT(5)
Number of function evaluations performed.

ITT(6)
Number of particles reset.

ITT(7)
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.

20: 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 | E05SBF 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.

21: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

*On exit*: the most common exit will be IFAIL $= 1$.

For this reason, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended; otherwise, the recommended value is $-1$. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

E05SBF returns IFAIL $= 0$ if and only if a finalization criterion has been reached which can guarantee success. This may only happen if:

(i) The option **Target Objective Value** has been set and has been reached at a sufficiently constrained point within the search domain.

(ii) The option **Optimize** = CONSTRAINTS has been set, and a sufficiently constrained point has been found within the search domain.

These finalization criteria are not active using default option settings, and must be explicitly set using E05ZKF if required.

E05SBF will return IFAIL $= 1$ if no error has been detected, and a finalization criterion has been achieved which cannot guarantee success. This does not indicate that the routine 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.

Other positive values of IFAIL indicate that either an error or a warning has been triggered. See Sections 6, 7 and 11 for more information.

## 6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or $-1$, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

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

IFAIL $= 2$

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 routine. 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 routine is called, it may imply that the objective is largely unimodal, and that it may be more efficient to use the routine 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**.

IFAIL $= 3$

User requested exit $\langle value \rangle$ during call to CONFUN.

User requested exit $\langle value \rangle$ during call to MONMOD.

User requested exit $\langle value \rangle$ during call to OBJFUN.

IFAIL $= 4$

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

IFAIL $= 11$

On entry, NDIM $= \langle value \rangle$.
Constraint: NDIM $\geq 1$.

IFAIL $= 12$

On entry, NPAR $= \langle value \rangle$.
Constraint: NPAR $\geq 5 \times$ **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 routine.

IFAIL $= 13$

On entry, NCON $= \langle value \rangle$.
Constraint: NCON $\geq 0$.

IFAIL $= 14$

On entry, BL($\langle value \rangle$) $= \langle value \rangle$ and BU($\langle value \rangle$) $= \langle value \rangle$.
Constraint: BU($i$) $\geq$ BL($i$) for all $i$.

On entry, BL($i$) $=$ BU($i$) for all box bounds $i$.
Constraint: BU($i$) $>$ BL($i$) for at least one box bound $i$.

IFAIL $= 17$

E05SBF has been called with NCON $> 0$ and the dummy constraint function E05SZM. Only use E05SZM with NCON $= 0$.

IFAIL $= 18$

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

IFAIL $= 19$

Error $\langle value \rangle$ occurred whilst adjusting to exterior local minimizer options.

Error $\langle value \rangle$ occurred whilst adjusting to interior local minimizer options.

IFAIL $= 21$

Either the option arrays have not been initialized for E05SBF, or they have become corrupted.

IFAIL $= 32$

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

IFAIL $= 51$

Multiple SMP threads have been detected; however, the option **SMP Callback Thread Safe** has not been set.
Set **SMP Callback Thread Safe** $=$ YES if the provided callbacks are thread safe.

Set **SMP Callback Thread Safe** = NO if the provided callbacks are not thread safe, to force serial execution.

IFAIL = −99

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL = −399

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL = −999

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

## 7 Accuracy

If IFAIL = 0 (or IFAIL = 2) or IFAIL = 1 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 E05SBF, 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, IFAIL = 0 (or IFAIL = 2). 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, IFAIL = 1 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

E05SBF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

E05SBF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

The algorithm has been parallelized to allow for a high degree of asynchronicity between threads. Each thread is assigned a static number of the NPAR particles requested, and performs a sub-iteration using these particles and a private copy of $\tilde{\mathbf{x}}$. The thread only updates this private copy if a superior solution is found. In these implementations, this routine may make calls to the user-supplied functions from within

an OpenMP parallel region. Thus OpenMP directives within the user functions can only be used if you are compiling the user-supplied function and linking the executable in accordance with the instructions in the Users' Note for your implementation.

Once a thread has completed a sub-iteration, it enters a brief critical section where it compares this private $\tilde{\mathbf{x}}$ to a globally accessible version. If either is superior, the inferior version is updated and the thread continues into a new sub-iteration.

Parallelizing the algorithm in this way allows for individual threads to continue searching even if other threads are completing sub-iterations in inferior times. The optional argument **SMP Thread Overrun** allows you to force a synchronization across the team of threads once one thread completes sufficiently more sub-iterations than the slowest thread. In particular, this may be used to force synchronization after every sub-iteration if so desired.

When using an SMP parallel version of this routine, you must indicate that the callback routines are thread safe by setting the optional argument **SMP Callback Thread Safe** before calling E05SBF in a multi-threaded environment. See Section 12.2 for more information on this and other SMP options.

**Note:** the stochastic method used in E05SBF will not produce repeatable answers when run on multiple threads.

## 9 Further Comments

The memory used by E05SBF 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, E05SBF 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 parameters 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.

(For SMP users only) The option **Repeatability** = ON will use preset seeds to initialize the random number generator on each thread, however due to the unpredictable nature of parallel communication, this cannot ensure repeatable results when running on multiple threads, even with **SMP Thread Overrun** set to force synchronization every iteration.

## 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:

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

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 E05ZKF, how to query options using E05ZLF, and finally how to search for the global optimum using E05SBF. The problem is solved twice, first using E05SBF alone, and secondly by coupling E05SBF with E04UCF/E04UCA as a

dedicated local minimizer. In both cases the default option **Repeatability** = ON is used to produce repeatable solutions.

**Note**: for users of multi-threaded implementations of the NAG Library the following example program does not include the setting of the optional parameter **SMP Callback Thread Safe**, and as such if run on multiple threads it will issue an error message. See the additional example program provided for E05SAF for more information on how to safely access independent subsections of the provided IUSER and RUSER arrays from multiple threads and how to use E05ZKF to set additional SMP threading related options.

## 10.1 Program Text

```
!   E05SBF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module e05sbfe_mod

!      E05SBF Example Program Module:
!             Parameters and User-defined Routines

!      NAG COPYRIGHT 2011.

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
      Integer, Parameter                   :: detail_level = 0, i_liuser = 1
      Integer, Parameter                   :: i_lruser = i_liuser + 1
      Integer, Parameter                   :: liopts = 100, lopts = 100,       &
                                              min_lruser = 0, nin = 5,         &
                                              nout = 6, report_freq = 100
      Integer, Parameter                   :: i_liuser_req = i_lruser + 1
      Integer, Parameter                   :: i_lruser_req = i_liuser_req + 1
      Integer, Parameter                   :: i_nlabels = i_lruser_req + 1
      Integer, Parameter                   :: i_nlengths = i_nlabels + 1
      Integer, Parameter                   :: i_max_threads = i_nlengths + 1
      Integer, Parameter                   :: i_ndim = i_max_threads + 1
      Integer, Parameter                   :: i_ncon = i_ndim + 1
      Integer, Parameter                   :: i_iuser_start = i_ncon + 1
      Integer, Parameter                   :: i_ruser_start = i_iuser_start + 1
      Integer, Parameter                   ::                                  &
                                  i_liuser_callback_shared = i_ruser_start + 1
      Integer, Parameter                   ::                                  &
                       i_liuser_callback_private = i_liuser_callback_shared + 1
      Integer, Parameter                   ::                                  &
                       i_lruser_callback_shared = i_liuser_callback_private + 1
      Integer, Parameter                   ::                                  &
                       i_lruser_callback_private = i_lruser_callback_shared + 1
      Integer, Parameter                   ::                                  &
                      i_liuser_callback_scratch = i_lruser_callback_private + 1
      Integer, Parameter                   ::                                  &
                     i_lruser_callback_scratch = i_liuser_callback_scratch + 1
      Integer, Parameter                   ::                                  &
                  i_label_iuser_callback_shared = i_lruser_callback_scratch + 1
      Integer, Parameter                   ::                                  &
             i_label_iuser_callback_private = i_label_iuser_callback_shared + 1
      Integer, Parameter                   ::                                  &
             i_label_ruser_callback_shared = i_label_iuser_callback_private + 1
      Integer, Parameter                   ::                                  &
             i_label_ruser_callback_private = i_label_ruser_callback_shared + 1
      Integer, Parameter                   ::                                  &
                    i_label_iuser_scratch = i_label_ruser_callback_private + 1
      Integer, Parameter                   ::                                  &
                           i_label_ruser_scratch = i_label_iuser_scratch + 1
      Integer, Parameter                   ::                                  &
                                     min_liuser = i_label_ruser_scratch + 1
    Contains
```

```
      Subroutine initialize_callback_arrays(ndim,ncon,iuser,liuser,ruser, &
        lruser,ifail)
!         Subroutine to ensure sufficient space in user arrays for labels and to
set
!         default user array values.

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (Out)                :: ifail
        Integer, Intent (In)                 :: liuser, lruser, ncon, ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(lruser)
        Integer, Intent (Inout)              :: iuser(liuser)
!       .. Local Scalars ..
        Integer                              :: max_threads, nlabels, nlengths
!       .. External Procedures ..
        Integer, External                    :: omp_get_max_threads
!       .. Executable Statements ..
        ifail = 0

        max_threads = 1
        !$ max_threads = omp_get_max_threads()

        nlabels = (ncon+1)*(max_threads+1)
        nlengths = 4*(ncon+1)
        iuser(3:liuser) = 0
        ruser(1:lruser) = 0.0E0_nag_wp

!       Set fixed data
        iuser(i_liuser) = liuser
        iuser(i_lruser) = lruser
        iuser(i_ndim) = ndim
        iuser(i_ncon) = ncon
        iuser(i_max_threads) = max_threads
        iuser(i_liuser_req) = min_liuser + nlabels + nlengths + 1
        iuser(i_lruser_req) = min_lruser + 2

!       Set references to length references.
        iuser(i_liuser_callback_shared) = min_liuser + 1
        iuser(i_liuser_callback_private) = iuser(i_liuser_callback_shared) + &
          ncon + 1
        iuser(i_lruser_callback_shared) = iuser(i_liuser_callback_private) + &
          ncon + 1
        iuser(i_lruser_callback_private) = iuser(i_lruser_callback_shared) + &
          ncon + 1

!         Set references to workspace references.
!         Require ncon+1 'shared' workpsace references.
        iuser(i_label_iuser_callback_shared) = iuser(i_lruser_callback_private &
          ) + ncon + 1
        iuser(i_label_ruser_callback_shared) = iuser( &
          i_label_iuser_callback_shared) + ncon + 1

!       Require max_threads*(ncon+1) 'private' workspace references.
        iuser(i_label_iuser_callback_private) = iuser( &
          i_label_ruser_callback_shared) + (ncon+1)
        iuser(i_label_ruser_callback_private) = iuser( &
          i_label_iuser_callback_private) + (ncon+1)*max_threads

!       Require max_threads 'scratch' workspace references
        iuser(i_label_iuser_scratch) = iuser(i_label_iuser_callback_private) + &
          (ncon+1)*max_threads
        iuser(i_label_ruser_scratch) = iuser(i_label_iuser_scratch) + &
          max_threads

!       Set references to first available workspace to be after last label
        iuser(i_iuser_start) = iuser(i_label_ruser_scratch) + max_threads + 1
        iuser(i_ruser_start) = min_lruser + 1

        Return
```

```
      End Subroutine initialize_callback_arrays
      Subroutine calculate_workspace(ndim,ncon,max_threads,min_liuser, &
        min_lruser,fileid,liuser_calc,lruser_calc,ifail)
!         Subroutine to calculate required sizes of iuser and ruser from
!         information in data file for individual objective and constraint
!         functions.
!         _shared indicates workspace that is available to all threads.
!         _private indicates workspace that is private to one thread,
!                  and maintained between function calls
!         _scratch indicates workspace that is private to each thread
!                  and not maintained between function calls

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: fileid, max_threads,          &
                                               min_liuser, min_lruser, ncon, &
                                               ndim
        Integer, Intent (Out)               :: ifail, liuser_calc, lruser_calc
!       .. Local Scalars ..
        Integer                             :: a, astat, b, c, file_funid,   &
                                               funid, liuser_private,        &
                                               liuser_scratch,              &
                                               liuser_scratch_max,          &
                                               liuser_shared, lruser_private, &
                                               lruser_scratch,              &
                                               lruser_scratch_max,          &
                                               lruser_shared
        Character (20)                      :: buffer
!       .. Intrinsic Procedures ..
        Intrinsic                           :: max
!       .. Executable Statements ..
        ifail = 0

!         Initialise to the required space for labels and storage of
!         parameters in iuser and ruser

        liuser_calc = min_liuser + (ncon+1)*(max_threads+6) + 2*max_threads + &
          3
        lruser_calc = min_lruser + 2

!         Load details of required workspace.
!         Rewind data file, then skip lines in data file until the
!         correct function or constraint data is found.

        Rewind (fileid)

        liuser_scratch_max = 0
        lruser_scratch_max = 0

        Do funid = 0, ncon

loop_data_set: Do
            Read (fileid,'(A20)',Iostat=astat) buffer

            If (astat<0) Then
!             End of file. Assume no data requried.
              Exit loop_data_set
            Else If (funid==0 .And. buffer(1:6)=='OBJFUN') Then
!             Heading for objective function detected.
              Exit loop_data_set
            Else If (buffer(1:6)=='CONFUN') Then
!             Heading for constraint detected.
              Read (buffer(7:20),*,Iostat=astat) file_funid
              If (funid==file_funid) Then
!               Correct constraint number detected.
                Exit loop_data_set
              End If
            End If
          End Do loop_data_set
```

```
          If (astat<0) Then
            liuser_shared = 0
            lruser_shared = 0
            liuser_private = 0
            lruser_private = 0
            liuser_scratch = 0
            lruser_scratch = 0
          Else
!             Read required memory parameters.
!             Here this is stored as a quadratic in ndim.
            Read (fileid,*) a, b, c
            liuser_shared = (a*ndim+b)*ndim + c
            Read (fileid,*) a, b, c
            liuser_private = (a*ndim+b)*ndim + c
            Read (fileid,*) a, b, c
            liuser_scratch = (a*ndim+b)*ndim + c
            Read (fileid,*) a, b, c
            lruser_shared = (a*ndim+b)*ndim + c
            Read (fileid,*) a, b, c
            lruser_private = (a*ndim+b)*ndim + c
            Read (fileid,*) a, b, c
            lruser_scratch = (a*ndim+b)*ndim + c
          End If

!         Each thread requires only 1 length of scratch workspace.
          liuser_scratch_max = max(liuser_scratch_max,liuser_scratch)
          lruser_scratch_max = max(lruser_scratch_max,lruser_scratch)

          liuser_calc = liuser_calc + max_threads*(liuser_private) + &
            liuser_shared
          lruser_calc = lruser_calc + max_threads*(lruser_private) + &
            lruser_shared

        End Do

        liuser_calc = liuser_calc + max_threads*liuser_scratch_max
        lruser_calc = lruser_calc + max_threads*lruser_scratch_max

        Return
      End Subroutine calculate_workspace
      Subroutine load_callback_data(funid,fileid,iuser,ruser,ifail)
!         Subroutine to load user data from the data file
!         for individual objective and constraint functions.
!         This should only be called when NSTATE = 2
!         funid = 0 if called in OBJFUN.
!         funid = constraint number if called in confun.
!         _shared indicates workspace that is available to all threads.
!         _private indicates workspace that is private to one thread,
!                  and maintained between function calls
!         _scratch indicates workspace that is private to each thread
!                  and not maintained between function calls

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: fileid, funid
        Integer, Intent (Out)               :: ifail
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)  :: ruser(*)
        Integer, Intent (Inout)             :: iuser(*)
!       .. Local Scalars ..
        Integer                             :: a, astat, b, c, file_funid, k, &
                                               labi_private, labi_shared,     &
                                               labr_private, labr_shared,     &
                                               lab_labi_private,              &
                                               lab_labi_shared,               &
                                               lab_labr_private,              &
                                               lab_labr_shared,               &
                                               liuser_private,                &
                                               liuser_scratch, liuser_shared, &
                                               lruser_private,                &
```

```
                                                  lruser_scratch, lruser_shared, &
                                                  max_threads, ncon, ndim
        Character (20)                         :: buffer
!       .. Executable Statements ..
        ifail = 0
        ndim = iuser(i_ndim)
        ncon = iuser(i_ncon)
        max_threads = iuser(i_max_threads)

!         Load details of required workspace.
!         Rewind data file, then skip lines in data file until the
!         correct function or constraint data is found.

        Rewind (fileid)
loop_data_set: Do
          Read (fileid,'(A20)',Iostat=astat) buffer

          If (astat<0) Then
!           End of file. Assume no data requried.
            Exit loop_data_set
          Else If (funid==0 .And. buffer(1:6)=='OBJFUN') Then
!           Heading for objective function detected.
            Exit loop_data_set
          Else If (buffer(1:6)=='CONFUN') Then
!           Heading for constraint detected.
            Read (buffer(7:20),*,Iostat=astat) file_funid
            If (funid==file_funid) Then
!             Correct constraint number detected.
              Exit loop_data_set
            End If
          End If
        End Do loop_data_set

        If (astat<0) Then
          liuser_shared = 0
          lruser_shared = 0
          liuser_private = 0
          lruser_private = 0
          liuser_scratch = 0
          lruser_scratch = 0
        Else
!           Read required memory parameters.
!           Here this is stored as a quadratic in ndim.
          Read (fileid,*) a, b, c
          liuser_shared = (a*ndim+b)*ndim + c
          Read (fileid,*) a, b, c
          liuser_private = (a*ndim+b)*ndim + c
          Read (fileid,*) a, b, c
          liuser_scratch = (a*ndim+b)*ndim + c
          Read (fileid,*) a, b, c
          lruser_shared = (a*ndim+b)*ndim + c
          Read (fileid,*) a, b, c
          lruser_private = (a*ndim+b)*ndim + c
          Read (fileid,*) a, b, c
          lruser_scratch = (a*ndim+b)*ndim + c
        End If

!         Each objective or constraint may have an assigned amount of
!         'shared' and 'private' workspace
        iuser(iuser(i_liuser_callback_shared)+funid) = liuser_shared
        iuser(iuser(i_lruser_callback_shared)+funid) = lruser_shared
        iuser(iuser(i_liuser_callback_private)+funid) = liuser_private
        iuser(iuser(i_lruser_callback_private)+funid) = lruser_private

!       Each thread requires only 1 length of scratch workspace.
        If (iuser(i_liuser_callback_scratch)<liuser_scratch) Then
          iuser(i_liuser_callback_scratch) = liuser_scratch
        End If
        If (iuser(i_lruser_callback_scratch)<lruser_scratch) Then
          iuser(i_lruser_callback_scratch) = lruser_scratch
        End If
```

```
!       Set up labels for shared workspace to the next available spaces.
        lab_labi_shared = iuser(i_label_iuser_callback_shared) + funid
        iuser(lab_labi_shared) = iuser(i_iuser_start)
        lab_labr_shared = iuser(i_label_ruser_callback_shared) + funid
        iuser(lab_labr_shared) = iuser(i_ruser_start)

!       Set up labels for private data for thread 0 (Master)
        lab_labi_private = iuser(i_label_iuser_callback_private) + &
          max_threads*funid
        labi_private = iuser(lab_labi_shared) + liuser_shared
        iuser(lab_labi_private) = labi_private

        lab_labr_private = iuser(i_label_ruser_callback_private) + &
          max_threads*funid
        labr_private = iuser(lab_labr_shared) + lruser_shared
        iuser(lab_labr_private) = labr_private

!         Set i_iuser_start and i_ruser_start to be after shared and private
!         space requried for individual function.
        iuser(i_iuser_start) = labi_private + max_threads*liuser_private
        iuser(i_ruser_start) = labr_private + max_threads*lruser_private

!         Adjust scratch workspace labels to point after all
!         required private and shared workspace
        iuser(iuser(i_label_iuser_scratch)) = iuser(i_iuser_start) + 1
        iuser(iuser(i_label_ruser_scratch)) = iuser(i_ruser_start) + 1

!         Ensure there is sufficient space allocated in iuser and ruser.
!         Note that as this is definitely in a serial region,
!         no critical section is required.
        iuser(i_liuser_req) = iuser(iuser(i_label_iuser_scratch)) + &
          iuser(i_liuser_callback_scratch)*max_threads
        iuser(i_lruser_req) = iuser(iuser(i_label_ruser_scratch)) + &
          iuser(i_lruser_callback_scratch)*max_threads

        If (iuser(i_liuser)<iuser(i_liuser_req) .Or. &
          iuser(i_lruser)<iuser(i_lruser_req)) Then
          ifail = -2
          Write (nout,99999) buffer(1:6)
          If (funid==0) Then
            Write (nout,99998) ncon, max_threads
          Else
            Write (nout,99998) funid, max_threads
          End If
          Write (nout,99997) iuser(i_liuser), iuser(i_liuser_req), &
            iuser(i_lruser), iuser(i_lruser_req)
          Write (nout,99996)
          Go To 100
        End If
        labi_shared = iuser(lab_labi_shared)
        labr_shared = iuser(lab_labr_shared)
!       Read in any shared data requried for iuser, if any.
        Do k = 1, liuser_shared
          Read (fileid,*) iuser(labi_shared+k-1)
        End Do
!       Read in any shared data required for ruser, if any.
        Do k = 1, lruser_shared
          Read (fileid,*) ruser(labr_shared+k-1)
        End Do

100     Continue
        Return

99999   Format (1X,'** Info: IUSER or RUSER is insufficient for ',A6, &
          ' when **')
99998   Format (1X,'** NCON >= ',I10,' and MAX_THREADS = ',I10,'.**')
99997   Format (1X,'** LIUSER = ',I10,'. Require LIUSER > ',I10,'.**'/1X, &
          '** LRUSER = ',I10,'. Require LRUSER > ',I10,'.**')
99996   Format (1X,'** The example program will now try to allocate &
          &sufficient space.**')
```

```
      End Subroutine load_callback_data
      Subroutine objfun_schwefel_objective(ndim,x,f,w)
!         subroutine to calculate the objective using provided workspace

!       .. Use Statements ..
        Use nag_library, Only: ddot
!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f
        Integer, Intent (In)                 :: ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: w(ndim)
        Real (Kind=nag_wp), Intent (In)      :: x(ndim)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: abs, sin, sqrt
!       .. Executable Statements ..
        w = abs(x)
        w = sqrt(w)
        w = sin(w)
        f = ddot(ndim,x,1,w,1)

        Return
      End Subroutine objfun_schwefel_objective
      Subroutine objfun_schwefel_gradient(ndim,x,g,w1,w2)
!         subroutine to calculate the gradient of the objective function
!         with provided workspace

!       .. Use Statements ..
        Use nag_library, Only: dscal
!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: g(ndim), w1(ndim), w2(ndim)
        Real (Kind=nag_wp), Intent (In)      :: x(ndim)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: abs, cos, sin, sqrt
!       .. Executable Statements ..
        Continue

        w1 = sqrt(abs(x))
        w2 = cos(w1)
        g = sin(w1)
        Call dscal(ndim,0.5E0_nag_wp,w1,1)
        g = g + w1*w2

        Return
      End Subroutine objfun_schwefel_gradient
      Subroutine objfun_schwefel(mode,ndim,x,objf,vecout,nstate,iuser,ruser)
!         Objfun routine for the Schwefel function for E05SBF_SMP.

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: objf
        Integer, Intent (Inout)              :: mode
        Integer, Intent (In)                 :: ndim, nstate
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*), vecout(ndim)
        Real (Kind=nag_wp), Intent (In)      :: x(ndim)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Local Scalars ..
        Integer                              :: ifail, labi_private,          &
                                                labi_scratch, labi_shared,    &
                                                labr_private, labr_scratch,   &
                                                labr_shared, lab_labi_private, &
                                                lab_labi_private_master,      &
                                                lab_labi_scratch,             &
```

```
                                                lab_labi_scratch_master,       &
                                                lab_labr_private,              &
                                                lab_labr_private_master,       &
                                                lab_labr_scratch,              &
                                                lab_labr_scratch_master,       &
                                                threadno
      Logical                                :: evalobjf, evalobjg
!     .. External Procedures ..
      Integer, External                      :: omp_get_thread_num
!     .. Intrinsic Procedures ..
      Intrinsic                              :: real
!     .. Executable Statements ..
!     Get the current thread number.
      threadno = 0
      !$ threadno = omp_get_thread_num()

!     Test NSTATE to indicate what stage of computation has been reached.
      Select Case (nstate)
      Case (2)

!          OBJFUN is called for the very first time.
!          This occurs in serial on thread 0 only.
        Call load_callback_data(0,nin,iuser,ruser,ifail)
        If (ifail/=0) Then
          mode = ifail
        End If

!       Perform any one time only calculations required.
        labr_shared = iuser(iuser(i_label_ruser_callback_shared))
        ruser(labr_shared) = real(ndim,nag_wp)*ruser(labr_shared)

      Case (3)
!          OBJFUN has been called for the first time on a new thread.

!          This occurs at the start of the main parallel region.
!          This will not occur on thread 0.

!          In this example we take the opportunity to set the labels for this
!          threads unique section of RUSER workspace in IUSER.
!          This could also be done when NSTATE=2
        lab_labi_private_master = iuser(i_label_iuser_callback_private)
        lab_labi_private = lab_labi_private_master + threadno
        labi_private = iuser(lab_labi_private_master) + &
          threadno*(iuser(iuser(i_liuser_callback_private)))
        iuser(lab_labi_private) = labi_private

        lab_labr_private_master = iuser(i_label_ruser_callback_private)
        lab_labr_private = lab_labr_private_master + threadno
        labr_private = iuser(lab_labr_private_master) + &
          threadno*(iuser(iuser(i_lruser_callback_private)))
        iuser(lab_labr_private) = labr_private

!       Set private scratch labels.
        lab_labi_scratch_master = iuser(i_label_iuser_scratch)
        lab_labi_scratch = lab_labi_scratch_master + threadno
        labi_scratch = iuser(lab_labi_scratch_master) + &
          threadno*iuser(i_liuser_callback_scratch)
        iuser(lab_labi_scratch) = labi_scratch

        lab_labr_scratch_master = iuser(i_label_ruser_scratch)
        lab_labr_scratch = lab_labr_scratch_master + threadno
        labr_scratch = iuser(lab_labr_scratch_master) + &
          threadno*iuser(i_lruser_callback_scratch)
        iuser(lab_labr_scratch) = labr_scratch

      Case (1)
!     OBJFUN is called on entry to a NAG local minimizer.
      Case (0)
!     This will be the normal value of NSTATE.
      Case Default
!          This should never happen, and indicates that an error has
```

```
!            occurred on the system. It is used here to demonstrate how
!            errors may be handled in a thread safe manner, in this case
!            by performing indicative IO in a critical section.
          mode = -1
          !$Omp Critical (e05sbfe_objfun_error)
            Write (nout,99999) threadno
          !$Omp End Critical (e05sbfe_objfun_error)
          Go To 100
        End Select

!       Test MODE to determine whether to calculate OBJF and/or OBJGRD.
        evalobjf = .False.
        evalobjg = .False.
        Select Case (mode)
        Case (0,5)
!         Only the value of the objective function is needed.
          evalobjf = .True.
        Case (1,6)
!         Only the values of the NDIM gradients are required.
          evalobjg = .True.
        Case (2,7)
!         Both the objective function and the NDIM gradients are required.
          evalobjf = .True.
          evalobjg = .True.
        End Select

!       Set labels for shared and private workspaces.
        labi_shared = iuser(iuser(i_label_iuser_callback_shared))
        lab_labi_private = iuser(i_label_iuser_callback_private) + threadno
        labi_private = iuser(lab_labi_private)

        labr_shared = iuser(iuser(i_label_ruser_callback_shared))
        lab_labr_private = iuser(i_label_ruser_callback_private) + threadno
        labr_private = iuser(lab_labr_private)

!       set labels for scratch workspaces.
        lab_labi_scratch = iuser(i_label_iuser_scratch) + threadno
        labi_scratch = iuser(lab_labi_scratch)
        lab_labr_scratch = iuser(i_label_ruser_scratch) + threadno
        labr_scratch = iuser(lab_labr_scratch)

        If (evalobjf) Then
!            Evaluate the objective function.
!            objf = const + sum( x*sin(abs(sqrt(x))))

!            Pass thread-safe workspace to objfun_schwefel_objective
          Call objfun_schwefel_objective(ndim,x,objf,ruser(labr_scratch))

!         Use value safely available to all threads.
          objf = objf + ruser(labr_shared)

        End If

        If (evalobjg) Then
!            Calculate the gradient of the objective function,
!            and return the result in VECOUT.
!            gradient = sin(sqrt(abs(x)))
!                       + sign(cos(sqrt(abs(x)))/2.0*sqrt(abs(x)),x)

          Call objfun_schwefel_gradient(ndim,x,vecout,ruser(labr_scratch), &
            ruser(labr_scratch+ndim))
        End If

100     Continue
        Return

99999   Format (1X,'** ERROR DETECTED : Thread number =',1X,I10)
      End Subroutine objfun_schwefel
      Subroutine confun_non_linear(mode,ncon,ndim,ldcj,needc,x,c,cjac,nstate, &
        iuser,ruser)
!         Subroutine used to supply constraints
```

```
!         Unless NCON = 0, this will always be
!         called before OBJFUN

!       .. Use Statements ..
        Use nag_library, Only: daxpy, dcopy, dscal, f06bmf, f06fjf, nag_wp
!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Integer, Intent (In)                :: ldcj, ncon, ndim, nstate
        Integer, Intent (Inout)             :: mode
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: c(ncon)
        Real (Kind=nag_wp), Intent (Inout)  :: cjac(ldcj,ndim), ruser(*)
        Real (Kind=nag_wp), Intent (In)     :: x(ndim)
        Integer, Intent (Inout)             :: iuser(*)
        Integer, Intent (In)                :: needc(ncon)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: l2xma, scal, sumsq
        Integer                             :: conid, ifail, labi_private,    &
                                               labi_scratch, labi_shared,     &
                                               labr_private, labr_scratch,    &
                                               labr_shared, lab_labi_private, &
                                               lab_labi_private_master,       &
                                               lab_labi_scratch,              &
                                               lab_labi_scratch_master,       &
                                               lab_labr_private,              &
                                               lab_labr_private_master,       &
                                               lab_labr_scratch,              &
                                               lab_labr_scratch_master,       &
                                               lruser_s, threadno
        Logical                             :: evalc, evalcjac
!       .. External Procedures ..
        Integer, External                   :: omp_get_thread_num
!       .. Executable Statements ..
        threadno = 0
        !$ threadno = omp_get_thread_num()

!       Test NSTATE to determine whether a new phase is beginning
        Select Case (nstate)
        Case (2)
!            Very first call to confun.
!            This occurs in serial on thread 0 only.

          Do conid = 1, ncon
!           Load details of required workspace for constraint k.
            Call load_callback_data(conid,nin,iuser,ruser,ifail)
            If (ifail/=0) Then
              mode = ifail
              Go To 100
            End If
          End Do

        Case (3)
!            First call to confun on a new thread.
!            Set labels to alloted workspace in user arrays.
          Do conid = 1, ncon
            lab_labi_private_master = iuser(i_label_iuser_callback_private) + &
              conid*iuser(i_max_threads)
            lab_labi_private = lab_labi_private_master + threadno

            labi_private = iuser(lab_labi_private_master) + &
              threadno*iuser(iuser(i_liuser_callback_private)+conid)
            iuser(lab_labi_private) = labi_private

            lab_labr_private_master = iuser(i_label_ruser_callback_private) + &
              conid*iuser(i_max_threads)
            lab_labr_private = lab_labr_private_master + threadno

            labr_private = iuser(lab_labr_private_master) + &
              threadno*iuser(iuser(i_lruser_callback_private)+conid)
            iuser(lab_labr_private) = labr_private
```

```
          End Do

!         Set private scratch labels.
          lab_labi_scratch_master = iuser(i_label_iuser_scratch)
          lab_labi_scratch = lab_labi_scratch_master + threadno
          labi_scratch = iuser(lab_labi_scratch_master) + &
            threadno*iuser(i_liuser_callback_scratch)
          iuser(lab_labi_scratch) = labi_scratch

          lab_labr_scratch_master = iuser(i_label_ruser_scratch)
          lab_labr_scratch = lab_labr_scratch_master + threadno
          labr_scratch = iuser(lab_labr_scratch_master) + &
            threadno*iuser(i_lruser_callback_scratch)
          iuser(lab_labr_scratch) = labr_scratch

        Case (1)
!         First call to confun at the start of a local minimzation.
!         Set any constant elements of the Jacobian matrix, if any.
        Case Default
!       Standard call to confun.
        End Select

!       MODE: are constraints, derivatives, or both are required?
        evalc = mode == 0 .Or. mode == 2
        evalcjac = mode == 1 .Or. mode == 2

!       Assign scratch workspace labels
        lab_labi_scratch = iuser(i_label_iuser_scratch) + threadno
        labi_scratch = iuser(lab_labi_scratch)

        lab_labr_scratch = iuser(i_label_ruser_scratch) + threadno
        labr_scratch = iuser(lab_labr_scratch)


loop_constraints: Do conid = 1, ncon
!         Only those for which needc is non-zero need be set.
          If (needc(conid)<=0) Then
            Cycle loop_constraints
          End If

!         Assign workspace labels.

          labi_shared = iuser(iuser(i_label_iuser_callback_shared)+conid)
          labr_shared = iuser(iuser(i_label_ruser_callback_shared)+conid)
          lruser_s = iuser(iuser(i_lruser_callback_shared)+conid)

          lab_labi_private = iuser(i_label_iuser_callback_private) + &
            iuser(i_max_threads)*conid + threadno
          labi_private = iuser(lab_labi_private)

          lab_labr_private = iuser(i_label_ruser_callback_private) + &
            iuser(i_max_threads)*conid + threadno
          labr_private = iuser(lab_labr_private)

          Select Case (conid)
          Case (1)
!           ||X - SHARED_DATA ||_2
            Call dcopy(ndim,x,1,ruser(labr_scratch),1)
            Call daxpy(lruser_s,-1.0E0_nag_wp,ruser(labr_shared),1, &
              ruser(labr_scratch),1)
            scal = 0.0E0_nag_wp
            sumsq = 1.0E0_nag_wp
            Call f06fjf(ndim,ruser(labr_scratch),1,scal,sumsq)
            l2xma = f06bmf(scal,sumsq)

            If (evalc) Then
              c(conid) = l2xma
!             update private constraint counter

              iuser(labi_private) = iuser(labi_private) + 1
!             safely update shared constraint counter
```

```
              !$Omp Critical (e05sbfe_smp_update_c_count)
                iuser(labi_shared) = iuser(labi_shared) + 1
              !$Omp End Critical (e05sbfe_smp_update_c_count)
            End If

            If (evalcjac) Then
              If (l2xma>0.0E0_nag_wp) Then
                l2xma = 1.0E0_nag_wp/l2xma
              End If
              Call dscal(ndim,l2xma,ruser(labr_scratch),1)
              Call dcopy(ndim,ruser(labr_scratch),1,cjac(conid,1),ldcj)
!             update private constraint jacobian counter
              iuser(labi_private+1) = iuser(labi_private+1) + 1
!             Safely update shared constraint jacobian counter
              !$Omp Critical (e05sbfe_smp_update_cjac_count)
                iuser(labi_shared+1) = iuser(labi_shared+1) + 1
              !$Omp End Critical (e05sbfe_smp_update_cjac_count)
            End If


          End Select



        End Do loop_constraints

!       If an immediate exit is required, return MODE<0
100     Continue
        Return
      End Subroutine confun_non_linear
      Subroutine monmod(ndim,ncon,npar,x,xb,fb,cb,xbest,fbest,cbest,itt,iuser, &
        ruser,inform)

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: fb
        Integer, Intent (Inout)              :: inform
        Integer, Intent (In)                 :: ncon, ndim, npar
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: cb(ncon), cbest(ncon,npar),    &
                                                fbest(npar), xb(ndim),         &
                                                xbest(ndim,npar)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*), x(ndim,npar)
        Integer, Intent (In)                 :: itt(7)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Local Scalars ..
        Integer                              :: fid, k, labi_con1_private,     &
                                                labi_con1_shared,              &
                                                lab_labi_con1_private,         &
                                                lab_labi_con1_shared
        Character (18)                       :: fname
        Character (3)                        :: leftno
!       .. Intrinsic Procedures ..
        Intrinsic                            :: adjustl, modulo, trim
!       .. Executable Statements ..
        If (detail_level>=2) Then
!         Report on the first iteration, and every report_freq iterations.
          If (itt(1)==1 .Or. modulo(itt(1),report_freq)==0 .And. inform<1000) &
            Then
!               To avoid using critical sections about IO,
!               allow each thread to open an output file.
!               Note, on entry INFORM = THREADNO.

            fid = 10 + inform

            lab_labi_con1_shared = iuser(i_label_iuser_callback_shared) + 1
            labi_con1_shared = iuser(lab_labi_con1_shared)
            lab_labi_con1_private = iuser(i_label_iuser_callback_private) + &
              iuser(i_max_threads)*1 + inform
            labi_con1_private = iuser(lab_labi_con1_private)
```

```
            Write (leftno,*) inform
            Write (fname,99995) trim(adjustl(leftno))
            Open (fid,File=fname,Position='APPEND')

            Write (fid,*) '* Iteration *', itt(1)
            Write (fid,*) '* Total function evaluations *', itt(5)
            Write (fid,*) '* Constraint 1 and Jacobian Evaluations *'
            Write (fid,*) '* Total :', iuser(labi_con1_shared), &
              iuser(labi_con1_private)
            Write (fid,*) '* Current global optimum candidate:'
            Do k = 1, ndim
              Write (fid,99999) k, xb(k)
            End Do
            Write (fid,*) '* Current global optimum value:'
            Write (fid,99998) fb
            Write (fid,99997)
            Do k = 1, ncon
              Write (fid,99996) k, cb(k)
            End Do
            Close (fid)
          End If
        End If

!       If required set INFORM<0 to force exit

        Return
99999   Format (1X,'* xb(',I3,') = ',F9.2)
99998   Format (1X,'* fb = ',F9.5)
99997   Format ('** Current global optimum constraint violations **')
99996   Format (1X,'* cb(',I3,') = ',F9.2)
99995   Format ('e05sbfe_smp_',A,'.r')
      End Subroutine monmod


      Subroutine display_result(ndim,ncon,xb,fb,cb,itt,inform)
!         Display final results in comparison to known global optimum.

!       .. Implicit None Statement ..
        Implicit None
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: fb
        Integer, Intent (In)                 :: inform, ncon, ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: cb(ncon), xb(ndim)
        Integer, Intent (In)                 :: itt(7)
!       .. Local Scalars ..
        Integer                              :: k
!       .. Executable Statements ..
!       Display final counters.
        Write (nout,*) ' Algorithm Statistics'
        Write (nout,*) ' --------------------'
        Write (nout,99992) 'Total complete iterations               ', itt(1)
        Write (nout,99992) 'Complete iterations since improvement   ', itt(2)
        Write (nout,99992) 'Total particles converged to xb         ', itt(3)
        Write (nout,99992) 'Total improvements to global optimum    ', itt(4)
        Write (nout,99992) 'Total function evaluations              ', itt(5)
        Write (nout,99992) 'Total particles re-initialized          ', itt(6)
        Write (nout,99992) 'Total constraints violated              ', itt(7)

!       Display why finalization occurred.
        Write (nout,*)
        Select Case (inform)
        Case (1)
          Write (nout,99999) 'Target value achieved'
        Case (2)
          Write (nout,99999) 'Minimum swarm standard deviation obtained'
        Case (3)
          Write (nout,99999) 'Sufficient particles converged'
        Case (4)
```

```
          Write (nout,99999) 'No improvement in preset iteration limit'
        Case (5)
          Write (nout,99999) 'Maximum complete iterations attained'
        Case (6)
          Write (nout,99999) 'Maximum function evaluations exceeded'
        Case (7)
          Write (nout,99999) 'Constrained point located'
        Case (:-1)
          Write (nout,99998) inform
        Case Default
        End Select

!       Display final objective value and location.
        Write (nout,*)
        Write (nout,99997) fb
        Write (nout,99996)
        Do k = 1, ndim
          Write (nout,99995) k, xb(k)
        End Do
        Write (nout,99994)
        Do k = 1, ncon
          Write (nout,99993) k, cb(k)
        End Do

        Write (nout,*)

        Return
99999   Format (2X,'Solution Status : ',A38)
99998   Format ('  User termination case                  : ',I13)
99997   Format ('  Achieved objective value               : ',F13.3)
99996   Format ('  Final position achieved  : ')
99995   Format ('  xb(',I4,') = ',F12.2)
99994   Format ('  Final constraint violations achieved  : ')
99993   Format ('  cb(',I4,') = ',F12.2)
99992   Format (2X,A40,' :',I13)

      End Subroutine display_result
    End Module e05sbfe_mod
    Program e05sbfe_smp
!      E05SBF SMP Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: e05sbf, e05zkf, e05zlf
      Use e05sbfe_mod, Only: calculate_workspace, confun_non_linear,          &
                             display_result, initialize_callback_arrays,      &
                             i_liuser_req, i_lruser_req, liopts, lopts,       &
                             min_liuser, min_lruser, monmod, nag_wp, nin,     &
                             nout, objfun_schwefel, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: fb, rvalue
      Integer                             :: astat, ifail, inform, ivalue, k, &
                                             liuser, lruser, max_threads,     &
                                             ncon, ndim, npar, optype
      Character (16)                      :: cvalue
      Character (80)                      :: optstr
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: bl(:), bu(:), cb(:), cbest(:,:), &
                                             fbest(:), ruser(:), xb(:),       &
                                             xbest(:,:)
      Real (Kind=nag_wp)                  :: opts(lopts)
      Integer                             :: iopts(liopts), itt(7)
      Integer, Allocatable                :: iuser(:)
!     .. External Procedures ..
      Integer, External                   :: omp_get_max_threads
!     .. Intrinsic Procedures ..
      Intrinsic                           :: max
!     .. Executable Statements ..
!     Print advisory information.
      Write (nout,*) 'E05SBF Example Program Results'
```

```
      Write (nout,*)
      Write (nout,*) 'Minimization of the Schwefel function.'
      Write (nout,*) 'Subject to one non-linear constraint.'
      Write (nout,*)

!     Skip heading in data file
      Read (nin,*)

!     Read parameters
      Read (nin,*) ndim
      Read (nin,*) ncon

      max_threads = 1
      !$ max_threads = omp_get_max_threads()

      npar = 10*max(max_threads,ndim)
      Allocate (xb(ndim),cb(ncon),xbest(ndim,npar),cbest(ncon,npar), &
        fbest(npar),bl(ndim+ncon),bu(ndim+ncon),Stat=astat)

      xbest = zero
      fbest = zero
      cbest = zero

!      Set problem specific values.
!      Set box bounds
      bl(1:ndim) = -500.0E0_nag_wp
      bu(1:ndim) = 500.0E0_nag_wp

!     Read general constraint bounds.
      Do k = 1, ncon
        Read (nin,*) bl(k+ndim), bu(k+ndim)
      End Do

!     Calculate size of LIUSER and LRUSER required.

      Call calculate_workspace(ndim,ncon,max_threads,min_liuser,min_lruser, &
        nin,liuser,lruser,ifail)

!     Initialize the option arrays for E05SBF.
      ifail = 0
      Call e05zkf('Initialize = E05SBF',iopts,liopts,opts,lopts,ifail)
!     ------------------------------------------------------------------
      Write (nout,*)
      Write (nout,*) '1. Solution without using coupled local minimizer'
      Write (nout,*)
!     Set various options to non-default values if required.
      ifail = 0
      Write (optstr,99999) 'Constraint Tolerance', 1.0E-5_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Constraint Superiority', 1.0E-6_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Constraint Norm = L2',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Repeatability = On',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Distance Tolerance', 1.0E-3_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Maximum Iterations Static = 1000',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Call e05zkf('Maximum Iterations Static Particles = 0',iopts,liopts,opts, &
        lopts,ifail)
      ifail = 0
      Call e05zkf('Constraint Warning = off',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Repulsion Initialize = 50',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Repulsion Finalize = 20',iopts,liopts,opts,lopts,ifail)
```

```
      ifail = 0
      Call e05zkf('Repulsion Particles = 1',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Boundary = Hyperspherical',iopts,liopts,opts,lopts,ifail)
      ifail = 0

!     It is essential to indicate callback thread safety.
      Call e05zkf('SMP Callback Thread Safe = Yes',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Call e05zkf('SMP Thread Overrun = 10',iopts,liopts,opts,lopts,ifail)
      ifail = 0

!            Call E05SBF to search for the global optimum.
!            This is incorporated into a loop so that, in the unlikely
!            event of insufficient space in iuser and ruser, the code
!            will automatically try to allocate sufficient space.

loop_no_min: Do

loop_initialize_user_arrays: Do
          Allocate (iuser(liuser),ruser(lruser),Stat=astat)

          Call initialize_callback_arrays(ndim,ncon,iuser,liuser,ruser,lruser, &
            ifail)
          If (ifail==0) Then
            Exit loop_initialize_user_arrays
          Else
            liuser = max(iuser(i_liuser_req),liuser)
            lruser = max(iuser(i_lruser_req),lruser)
          End If

          Deallocate (iuser,ruser)
        End Do loop_initialize_user_arrays


!       Non-zero IFAIL expected on exit here, so use IFAIL=1 (quiet) on entry.
        ifail = 1
        Call e05sbf(ndim,ncon,npar,xb,fb,cb,bl,bu,xbest,fbest,cbest, &
          objfun_schwefel,confun_non_linear,monmod,iopts,opts,iuser,ruser,itt, &
          inform,ifail)

!       It is essential to test IFAIL on exit.
        Select Case (ifail)
        Case (0,1)
!         No errors, best found optimum at xb returned in fb.
          Call display_result(ndim,ncon,xb,fb,cb,itt,inform)
        Case (3)
!         Exit flag set in OBJFUN, CONFUN or MONMOD and returned in INFORM.
          Select Case (inform)
          Case (-2)
!           Insufficient memory ineither iuser or ruser
            liuser = max(iuser(i_liuser_req),liuser)
            lruser = max(iuser(i_lruser_req),lruser)
            Deallocate (iuser,ruser)
            Cycle loop_no_min
          Case Default
            Call display_result(ndim,ncon,xb,fb,cb,itt,inform)
          End Select
        Case Default
!         An error was detected. Print message since IFAIL=1 on entry.
          Write (nout,99998) '** E05SBF returned with an error, IFAIL = ', &
            ifail
          Continue
        End Select
        Exit loop_no_min
      End Do loop_no_min

!      Deallocate user arrays. We will re-initialise them for the second
!      solution below.
      Deallocate (iuser,ruser)
```

```
!     ----------------------------------------------------------------
      Write (nout,*) '2. Solution using coupled local minimizer E04UCF'
      Write (nout,*)

!     Set the local minimizer to be E04UCF and set corresponding options.
      ifail = 0
      Call e05zkf('Local Minimizer = E04UCF',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Local Interior Major Iterations = 15',iopts,liopts,opts, &
        lopts,ifail)
      ifail = 0
      Call e05zkf('Local Interior Minor Iterations = 5',iopts,liopts,opts, &
        lopts,ifail)
      ifail = 0
      Call e05zkf('Local Exterior Major Iterations = 50',iopts,liopts,opts, &
        lopts,ifail)
      ifail = 0
      Call e05zkf('Local Exterior Minor Iterations = 15',iopts,liopts,opts, &
        lopts,ifail)

!     Query the option Distance Tolerance
      ifail = 0
      optstr = 'Distance Tolerance'
      Call e05zlf(optstr,ivalue,rvalue,cvalue,optype,iopts,opts,ifail)
!     Adjust Distance Tolerance dependent upon its current value
      Write (optstr,99999) 'Distance Tolerance', rvalue*10.0_nag_wp
      ifail = 0
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Local Interior Tolerance', rvalue
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Local Exterior Tolerance', rvalue*1.0E-4_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)


!               Call E05SBF to search for the global optimum using coupled
!               local minimizer E04UCF.
!               This is incorporated into a loop so that, in the unlikely
!               event of insufficient space in iuser and ruser, the code
!               will automatically try to allocate sufficient space.

loop_min_e04ucf: Do

loop_initialize_user_arrays_2: Do
          Allocate (iuser(liuser),ruser(lruser),Stat=astat)

          Call initialize_callback_arrays(ndim,ncon,iuser,liuser,ruser,lruser, &
            ifail)
          If (ifail==0) Then
            Exit loop_initialize_user_arrays_2
          Else
            liuser = max(iuser(i_liuser_req),liuser)
            lruser = max(iuser(i_lruser_req),lruser)
          End If

          Deallocate (iuser,ruser)
        End Do loop_initialize_user_arrays_2

!       Non-zero IFAIL expected on exit here, so use IFAIL=1 (quiet) on entry.
        ifail = 1
        Call e05sbf(ndim,ncon,npar,xb,fb,cb,bl,bu,xbest,fbest,cbest, &
          objfun_schwefel,confun_non_linear,monmod,iopts,opts,iuser,ruser,itt, &
          inform,ifail)

!       It is essential to test IFAIL on exit.
        Select Case (ifail)
        Case (0,1)
!         No errors, best found optimum at xb returned in fb.
          Call display_result(ndim,ncon,xb,fb,cb,itt,inform)
```

```
        Case (3)
!         Exit flag set in OBJFUN, CONFUN or MONMOD and returned in INFORM.
          Select Case (inform)
          Case (-2)
!           Insufficient memory ineither iuser or ruser
            liuser = max(iuser(i_liuser_req),liuser)
            lruser = max(iuser(i_lruser_req),lruser)
            Deallocate (iuser,ruser)
            Cycle loop_min_e04ucf
          Case Default
            Call display_result(ndim,ncon,xb,fb,cb,itt,inform)
          End Select
        Case Default
!         An error was detected. Print message since IFAIL=1 on entry.
          Write (nout,99998) '** E05SBF returned with an error, IFAIL = ', &
            ifail
          Continue
        End Select
        Exit loop_min_e04ucf
      End Do loop_min_e04ucf

99999 Format (A,' = ',E32.16)
99998 Format (1X,A,I6)
    End Program e05sbfe_smp
```

## 10.2 Program Data

None.

## 10.3 Program Results

```
 E05SBF Example Program Results

 Minimization of the Schwefel function.
 Subject to one non-linear constraint.


 1. Solution without using coupled local minimizer

  Algorithm Statistics
  --------------------
  Total complete iterations               :         2696
  Complete iterations since improvement   :         1002
  Total particles converged to xb         :            0
  Total improvements to global optimum    :         1843
  Total function evaluations              :       593321
  Total particles re-initialized          :            0
  Total constraints violated              :            0

  Solution Status : No improvement in preset iteration lim

  Achieved objective value                :     -9191.499
  Final position achieved  :
  xb(   1) =      -406.08
  xb(   2) =      -406.75
  xb(   3) =      -406.74
  xb(   4) =      -404.31
  xb(   5) =      -405.53
  xb(   6) =      -407.49
  xb(   7) =      -406.80
  xb(   8) =      -407.76
  xb(   9) =       384.69
  xb(  10) =       384.64
  xb(  11) =       385.73
  xb(  12) =       386.92
  xb(  13) =       385.67
  xb(  14) =       386.42
  xb(  15) =       384.38
  xb(  16) =       385.73
  xb(  17) =       383.65
```

```
 xb(  18) =       387.38
 xb(  19) =        -1.55
 xb(  20) =      -406.73
 Final constraint violations achieved  :
 cb(   1) =         0.00

2. Solution using coupled local minimizer E04UCF

 Algorithm Statistics
 --------------------
 Total complete iterations               :         1046
 Complete iterations since improvement   :         1002
 Total particles converged to xb         :            0
 Total improvements to global optimum    :          117
 Total function evaluations              :       231946
 Total particles re-initialized          :            0
 Total constraints violated              :            0

 Solution Status : No improvement in preset iteration lim

 Achieved objective value                :     -9198.430
 Final position achieved  :
 xb(   1) =      -406.65
 xb(   2) =      -406.65
 xb(   3) =      -406.65
 xb(   4) =      -406.65
 xb(   5) =      -406.65
 xb(   6) =      -406.65
 xb(   7) =      -406.65
 xb(   8) =      -406.65
 xb(   9) =       385.51
 xb(  10) =       385.51
 xb(  11) =       385.51
 xb(  12) =       385.51
 xb(  13) =       385.51
 xb(  14) =       385.51
 xb(  15) =       385.51
 xb(  16) =       385.51
 xb(  17) =       385.51
 xb(  18) =       385.51
 xb(  19) =        -2.06
 xb(  20) =      -406.65
 Final constraint violations achieved  :
 cb(   1) =         0.00
```

## 11 Algorithmic Details

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

INITIALIZE for $j = 1, n_p$

$\mathbf{x}_j = \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$

$\hat{\mathbf{x}}_j = \begin{cases} \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}}) & \textbf{Start} = \text{COLD} \\ \hat{\mathbf{x}}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$

$\mathbf{v}_j = \mathbf{R} \in U(-\mathbf{V}_{\max}, \mathbf{V}_{\max})$

$\hat{f}_j = \begin{cases} F(\hat{\mathbf{x}}_j) & \textbf{Start} = \text{COLD} \\ \hat{f}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$

$\hat{\mathbf{e}}_j = \begin{cases} \mathbf{e}(\hat{\mathbf{x}}_j) & \textbf{Start} = \text{COLD} \\ \hat{\mathbf{e}}_j^0 & \textbf{Start} = \text{WARM} \end{cases}$

$w_j = \begin{cases} W_{\max} & \textbf{Weight Initialize} = \text{MAXIMUM} \\ W_{\text{ini}} & \textbf{Weight Initialize} = \text{INITIAL} \\ R \in U(W_{\min}, W_{\max}) & \textbf{Weight Initialize} = \text{RANDOMIZED} \end{cases}$

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 $j = 1, n_p$

$\mathbf{x}_j = \text{BOUNDARY}(\mathbf{x}_j, \boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$

$f_j = F(\mathbf{x}_j)$

$\mathbf{e}_j = \mathbf{e}(\mathbf{x}_j)$

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 $j = 1, n_p$

$\mathbf{v}_j = w_j\mathbf{v}_j + C_s\mathbf{D}_1(\hat{\mathbf{x}}_j - \mathbf{x}_j) + C_g\mathbf{D}_2(\tilde{\mathbf{x}} - \mathbf{x}_j)$

$\mathbf{x}_j = \mathbf{x}_j + \mathbf{v}_j$

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.

$n_p$ 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

$e_k(\mathbf{x})$ $k$th (scaled) constraint violation at $\mathbf{x}$, evaluated as $\min(c_k(\mathbf{x}) - l_{\mathrm{NDIM}+k}, 0.0) + \max(c_k(\mathbf{x}) - u_{\mathrm{NDIM}+k}, 0.0)$; this may be scaled by the maximum $k$th constraint found thus far

$\mathbf{e}(\mathbf{x})$ the array of NCON constraint violations, $e_k(\mathbf{x})$, for $k = 1, 2, \ldots, \mathrm{NCON}$, 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

$f_{\mathrm{scale}}$ 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

$\mathbf{R} \in U(\boldsymbol{\ell}_{\mathrm{box}}, \mathbf{u}_{\mathrm{box}})$
an array of random numbers whose $i$-th element is drawn from a uniform distribution in the range $(\boldsymbol{\ell}_{\mathrm{box}\,i}, \mathbf{u}_{\mathrm{box}\,i})$, for $i = 1, 2, \ldots, \mathrm{NDIM}$

$O_i$ 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

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 ≥ r_p and r_i ≤ I_s ≤ r_i + r_f)
          LOCMIN (x̃, f̃, ẽ, O_i)
          use −C_g instead of C_g in velocity updates
      if  (I_s = r_i + r_f)
          I_s = 0
```

## 12 Optional Parameters

This section can be skipped if you wish to use the default values for all optional parameters, otherwise, the following is a list of the optional parameters available and a full description of each optional parameter 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**
**SMP Callback Thread Safe**
**SMP Local Minimizer External**
**SMP Monitor**
**SMP Monmod**
**SMP Subswarm**
**SMP Thread Overrun**
**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 Parameters

For each option, we give a summary line, a description of the optional parameter 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 $\epsilon$ is a generic notation for ***machine precision*** (see X02AJF), and $Imax$ represents the largest representable integer value (see X02BBF).

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 E05SBF the maximum length of the parameter CVALUE used by E05ZLF 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 E05SBF 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 $\mathrm{BL}(i) \neq \mathrm{BU}(i)$, $i = 1, 2, \ldots, \mathrm{NDIM}$.

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, $\|\mathbf{e}\|_1 = \frac{1}{\mathrm{NCON}} \sum_{1}^{\mathrm{NCON}} |e_k|$

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

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

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

**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, IFAIL $= 4$. It is advisable to deactivate this option if IFAIL $= 0$ on entry and the satisfaction of all constraints is not program critical.

OFF

No error will be returned.

ON

An error 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 parameter defines $\epsilon_r$, 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 $\epsilon_r$ should reflect the relative precision of $1 + |F(\mathbf{x})|$; i.e., $\epsilon_r$ 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 $\epsilon_r$ 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 $\epsilon_r$ would be $10^{-10}$. The choice of $\epsilon_r$ 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 $\epsilon_r$ 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

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

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** | **Parameter/option** | **Default Interior** | **Default Exterior** |
|---|---|---|---|
| E04CBF | MAXCAL | NDIM + 10 | 2 × NDIM + 15 |
| E04DGF/E04DGA | **Iteration Limit** | max(30, 3 × NDIM) | max(50, 5 × NDIM) |
| E04UCF/E04UCA | **Major Iteration Limit** | max(10, 2 × NDIM) | max(30, 3 × NDIM) |

Unless set, these are functions of the parameters passed to E05SBF.

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.

**Note**: currently E04JYF or E04KZF are restricted to using 400 × NDIM and 50 × NDIM as function evaluation limits respectively. This applies to both local minimizations inside and outside the main loop. They may still be deactivated in either phase by setting $i = 0$, and may subsequently be reactivated in either phase by setting $i > 0$.

*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** | **Parameter/option** | **Default Interior** | **Default Exterior** |
|---|---|---|---|
| E04UCF/E04UCA | **Minor Iteration Limit** | $\max(10, 2 \times \text{NDIM})$ | $\max(30, 3 \times \text{NDIM})$ |

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

**Local Minimizer** $a$ Default = OFF

Allows for a choice of Chapter E04 routines 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.

E04CBF
: Use E04CBF as the local minimizer. This does not require the calculation of derivatives.

On a call to OBJFUN during a local minimization, MODE = 5.

E04KZF
: Use E04KZF as the local minimizer. This requires the calculation of derivatives in OBJFUN, as indicated by MODE.

The box bounds forwarded to this routine from E05SBF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SBF.

Accurate derivatives must be provided to this routine, and will not be approximated internally. Each iteration of this local minimizer also requires the calculation of both the objective function and its derivative. Hence on a call to OBJFUN during a local minimization, MODE = 7.

E04JYF
: Use E04JYF as the local minimizer. This does not require the calculation of derivatives.

On a call to OBJFUN during a local minimization, MODE = 5.

The box bounds forwarded to this routine from E05SBF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SBF.

E04DGF
E04DGA
: Use E04DGA 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.

E04UCF
E04UCA
: Use E04UCA 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 routine from E05SBF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SBF.

**Maximum Function Evaluations** $i$ Default $= Imax$

The maximum number of evaluations of the objective function. When reached this will return IFAIL $= 1$ 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 E05SBF will exit with IFAIL $= 1$ and INFORM $= 5$.

Unless set, this adapts to the parameters passed to E05SBF.

*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 E05SBF will exit with IFAIL $= 1$ 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 routine may exit due to **Maximum Iterations Static** with IFAIL $= 1$ and INFORM $= 4$.

*Constraint*: $i \geq 0$.

**Maximum Particles Converged** $i$ Default $= Imax$

The maximum number of particles that may converge to the current optimum. When achieved, E05SBF will exit with IFAIL $= 1$ 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 $= Imax$

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 $\left|\mathbf{v}_j\right| \leq r \times \left(\mathbf{u}_j - \boldsymbol{\ell}_j\right) = \mathbf{V}_{\max}$. 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 routine 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, E05SBF will exit with IFAIL $= 0$ 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 E05SBF. **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 routine.

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, E05SBF will exit with IFAIL $= 1$ 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.

In SMP parallel implementations of E05SBF, the standard deviation will be calculated based only on the particles local to the particular thread that checks for finalization. Considerably fewer particles may be used in this calculation than when the algorithm is run in serial. It is therefore recommended that you provide a smaller value of **Swarm Standard Deviation** when running in parallel than when running in serial.

*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) E05SBF will exit with IFAIL $= 0$ 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, E05SBF will exit with IFAIL $= 2$. This option may take any real value $r$, or the character ON/OFF as well as DEFAULT. If this option is queried using E05ZLF, 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 $r$, E05SBF will exit successfully with IFAIL $= 0$ 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 parameter **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, IFAIL $= 2$.

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

ON
: An error will be returned if the target objective is reached prematurely, and the routine will exit with IFAIL $= 2$.

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

## 12.2 Description of the SMP optional parameters

This section details additional options available to users of multi-threaded implementations of the NAG Library. In particular it includes the option **SMP Callback Thread Safe**, which must be set before calling E05SBF with multiple threads.

**SMP Callback Thread Safe** $a$ Default = WARNING

Declare that the callback routines you provide are or are not thread safe. In particular, this indicates that access to the shared memory arrays IUSER and RUSER from within your provided callbacks is done in

a thread safe manner. If these arrays are just used to pass constant data, then you may assume they are thread safe. If these are also used for workspace, or passing variable data such as random number generator seeds, then you must ensure these are accessed and updated safely. Whilst this can be done using OpenMP critical sections, we suggest their use is minimized to prevent unnecessary bottlenecks, and that instead individual threads have access to independent subsections of the provided arrays where possible.

YES
: The callback routines have been programmed in a thread safe way. The algorithm will use `OMP_NUM_THREADS` threads.

NO
: The callback routines are not thread safe. Setting this option will force the algorithm to run on a single thread only, and is advisable only for debugging purposes, or if you wish to parallelize your callback functions.

WARNING
: This will cause an immediate exit from E05SBF with IFAIL $= 51$ if multiple threads are detected. This is to inform you that you have not declared the callback functions either to be thread safe, or that they are thread unsafe and you wish the algorithm to run in serial.

**SMP Local Minimizer External** $a$ Default = ALL

Determines how many threads will attempt to locally minimize the best found solution after the routine has exited the main loop.

MASTER
: Only the master thread will attempt to find any improvement. The local minimization will be launched from the best known solution. All other threads will remain effectively idle.

ALL
: The master thread will perform a local minimization from the best known solution, while all other threads will perform a local minimization from randomly generated perturbations of the best known solution, increasing the chance of an improvement. Assuming all local minimizations will take approximately the same amount of computation, this will be effectively free in terms of real time. It will however increase the number of function evaluations performed.

**SMP Monitor** $a$ Default = SINGLE
**SMP Monmod** $a$

Determines whether the user-supplied function MONMOD is invoked once every sub-iteration each thread performs, or only once by a single thread after all threads have completed at least one sub-iteration.

SINGLE
: Only one thread will invoke MONMOD, after all threads have performed at least one sub-iteration.

ALL
: Each thread will invoke MONMOD each time it completes a sub-iteration. If you wish to alter X using MONMOD you should use this option, as MONMOD will only receive the arrays X, XBEST, FBEST and CBEST private to the calling thread.

**SMP Subswarm** $i$ Default = 1

Determines how many threads support a particle subswarm. This is an extra collection of particles constrained to search only within a hypercube of edge length $10.0 \times$ **Distance Tolerance** of the best point known to an individual thread. This may improve the number of iterations required to find a provided target, particularly if no local minimizer is in use.

If $i \leq 0$, then this will be disabled on all the threads.

If $i \geq$ `OMP_NUM_THREADS`, then all the threads will support a particle subswarm.

**SMP Thread Overrun** $i$ Default $= Imax$

This option provides control over the level of asynchronicity present in a simulation. In particular, a barrier synchronization between all threads is performed if any thread completes $i$ sub-iterations more than the slowest thread, causing all threads to be exposed to the current best solution. Allowing asynchronous behaviour does however allow individual threads to focus on different global optimum candidates some of the time, which can inhibit convergence to unwanted sub-optima. It also allows for threads to continue searching when other threads are completing sub-iterations at a slower rate.

If $i < 1$, then the algorithm will force a synchronization between threads at the end of each iteration.

---