NAG Library Routine Document

E05SAF

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

E05SAF is designed to search for the global minimum or maximum of an arbitrary function, using Particle Swarm Optimization (PSO). Derivatives are not required, although these may be used by an accompanying local minimization routine if desired. E05SAF is essentially identical to E05SBF, but with a simpler interface and with various optional parameters removed; otherwise most parameters are identical. In particular, E05SAF does not handle general constraints.

2 Specification

```
SUBROUTINE E05SAF (NDIM, NPAR, XB, FB, BL, BU, OBJFUN, MONMOD, IOPTS,      &
                   OPTS, IUSER, RUSER, ITT, INFORM, IFAIL)
INTEGER            NDIM, NPAR, IOPTS(*), IUSER(*), ITT(6), INFORM,         &
                   IFAIL
REAL (KIND=nag_wp) XB(NDIM), FB, BL(NDIM), BU(NDIM), OPTS(*), RUSER(*)
EXTERNAL           OBJFUN, MONMOD
```

Before calling E05SAF, E05ZKF **must** be called with OPTSTR set to '`Initialize = e05saf`'. Optional parameters may also be specified by calling E05ZKF before the call to E05SAF.

3 Description

E05SAF 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. 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:

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

where the objective $F(\mathbf{x})$ is a scalar function, $\mathbf{x}$ is a vector in R^{ndim} and the vectors $\boldsymbol{\ell} \leq \mathbf{u}$ are lower and upper bounds respectively for the $ndim$ variables. The objective function may be nonlinear. Continuity of F 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 E05SAF.

For multi-modal functions for which derivatives cannot be provided, particularly functions with a significant level of noise in their evaluation, E05SAF should be used either alone, or coupled with E04CBF.

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.

E05SAF may also be used to maximize the objective function (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, E05SAF 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 and OBJFUN.

E05SBF provides a comprehensive interface, allowing for 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 ≥ 1.

2: NPAR – INTEGER *Input*

On entry: $npar$, the number of particles to be used in the swarm. Assuming all particles remain within bounds, 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.

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

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

4: FB – REAL (KIND=nag_wp) *Output*

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

5: BL(NDIM) – REAL (KIND=nag_wp) array *Input*
6: BU(NDIM) – REAL (KIND=nag_wp) array *Input*

On entry: BL is $\boldsymbol{\ell}$, the array of lower bounds, BU is **u**, the array of upper bounds. The 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).

If $\mathrm{BL}(i) = \mathrm{BU}(i)$ for any $i \in \{1, \ldots, \mathrm{NDIM}\}$, variable i will remain locked to $\mathrm{BL}(i)$ regardless of the **Boundary** option selected.

It is strongly advised that you place sensible lower and upper bounds on all variables, even if your model allows for variables to be unbounded (using the option **Boundary** = ignore) since these define the initial search space.

Constraints:

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

7: 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 E05SAF 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 value of $F(\mathbf{x})$ found so far by a given particle. 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 E05SAF. 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 E05SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: 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 E05SXM. (E05SXM is included in the NAG Library) .

The specification of MONMOD is:

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

INTEGER            NDIM, NPAR, ITT(6), IUSER(*), INFORM
REAL (KIND=nag_wp) X(NDIM,NPAR), XB(NDIM), FB,                        &
                   XBEST(NDIM,NPAR), FBEST(NPAR), RUSER(*)
```

1: NDIM – INTEGER *Input*

On entry: the number of dimensions.

2: NPAR – INTEGER *Input*

On entry: the number of particles.

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

Note: the ith component of the jth particle, $x_j(i)$, is stored in 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.

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

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

5: FB – REAL (KIND=nag_wp) *Input*

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

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

Note: the ith component of the position of the jth 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).

7: 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.

8: ITT(6) – INTEGER array *Input*

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

9: IUSER$(*)$ – INTEGER array *User Workspace*

10: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

MONMOD is called with the parameters IUSER and RUSER as supplied to E05SAF. You are free to use the arrays IUSER and RUSER to supply information to MONMOD as an alternative to using COMMON global variables.

11: 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 E05SAF. 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 E05SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: 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 E05SAF, E05ZKF or E05ZLF.

10: 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 E05SAF, E05ZKF or E05ZLF.

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

IUSER is not used by E05SAF, but is passed directly to OBJFUN 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 E05SAF 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.

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

RUSER is not used by E05SAF, but is passed directly to OBJFUN 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 E05SAF 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.

13: ITT(6) – INTEGER array *Output*

On exit: integer iteration counters for E05SAF.

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.

14: 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	E05SAF 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.

15: 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.**

E05SAF will return IFAIL = 0 if and only if a finalization criterion has been reached which can guarantee success. This may only happen if the option **Target Objective Value** has been set and reached at a point within the search domain. The finalization criterion **Target Objective Value** is not activated using default option settings, and must be explicitly set using E05ZKF if required.

E05SAF 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 MONMOD.

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

IFAIL $= 11$

On entry, NDIM $= \langle value \rangle$.
Constraint: NDIM ≥ 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 $= 14$

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 i.
Constraint: BU(i) $>$ BL(i) for at least one i.

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 E05SAF, 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, either a **Target Objective Value** or finalization criterion has 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 E05SAF, 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 basin 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 $= 0$. This may only happen if a **Target Objective Value** is assigned and is reached by the algorithm.

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

E05SAF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

E05SAF 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 E05SAF in a multi-threaded environment. See Section 12.2 for more information on this and other SMP options.

Note: the stochastic method used in E05SAF will not produce repeatable answers when run on multiple threads.

9 Further Comments

The memory used by E05SAF is relatively static throughout. As such, E05SAF 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 Schwefel function:

$$\underset{\mathbf{x}\in R^{ndim}}{\text{minimize}} f = \sum_{i=1}^{ndim} x_i \sin\left(\sqrt{|x_i|}\right)$$

$$x_i \in (-500, 500), \quad \text{for } i = 1, 2, \ldots, \text{NDIM}.$$

In two dimensions the optimum is $f_{\min} = -837.966$, located at $\mathbf{x} = (-420.97, -420.97)$.

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 E05SAF. The function is minimized several times to demonstrate using E05SAF alone, and coupled with local minimizers. This program uses the non-default option **Repeatability** = ON to produce repeatable solutions.

Note: for users of multi-threaded implementations of the NAG Library 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. An additional example program, `e05safe_smp.f90`, is included with the distribution material of all implementations of multi-threaded implementations of the NAG Library to illustrate 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

```
!   E05SAF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module e05safe_mod

!     E05SAF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter        ::                                 &
                                     f_target = -4.189828872724337E2_nag_wp
      Real (Kind=nag_wp), Parameter        :: one = 1.0E0_nag_wp
      Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
      Integer, Parameter                   :: detail_level = 0,               &
                                              iuser_reference = 1,            &
                                              liopts = 100, lopts = 100,      &
                                              ndim = 20, nout = 6,            &
                                              report_freq = 100,              &
                                              ruser_global = 1
      Real (Kind=nag_wp), Parameter        ::                                 &
                                  x_target(1:ndim) = -420.9687463599820_nag_wp
    Contains
      Subroutine objfun_schwefel_objective(ndim,x,f,w)
!       subroutine to calculate the objective using provided workspace

!       .. Use Statements ..
        Use nag_library, Only: ddot
!       .. 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
!       .. 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 E05SAF.

!       .. 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                             :: threadno, work_1, work_2
        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.

!         Set up any constant global variables required in RUSER and IUSER
!         This may also be done prior to calling E05SAF
          ruser(ruser_global) = -f_target*real(ndim,nag_wp)

!         Set reference index for thread 0 workspace in IUSER
          iuser(iuser_reference+threadno) = ruser_global + 1

        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 reference index for
this
!         threads unique section of RUSER workspace in IUSER.

!         Each thead requires 2*ndim real numbers.
          iuser(iuser_reference+threadno) = iuser(iuser_reference) + &
            2*ndim*threadno

        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 is extremely unlikely, and indicates that an error has
!         occurred on the system
          mode = -1
          If (detail_level>=2) Then
            Write (nout,99999)
          End If
          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.
        Case Default
          mode = -1
          Write (nout,99999)
          Go To 100
        End Select

!       Set work_1 and work_2 to point to the section of RUSER reserved
!       for this thread
        work_1 = iuser(iuser_reference+threadno)
        work_2 = work_1 + ndim

        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(work_1))
!         Use constant safely available to all threads.
          objf = objf + ruser(ruser_global)

        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(work_1), &
            ruser(work_2))
        End If

100     Continue
        Return

99999   Format (1X,'** ERROR DETECTED ')

      End Subroutine objfun_schwefel
      Subroutine monmod(ndim,npar,x,xb,fb,xbest,fbest,itt,iuser,ruser,inform)
!       Example monitor function.

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: fb
        Integer, Intent (Inout)              :: inform
        Integer, Intent (In)                 :: ndim, npar
!       .. Array Arguments ..
```

```
        Real (Kind=nag_wp), Intent (In)      :: fbest(npar), xb(ndim),         &
                                                xbest(ndim,npar)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*), x(ndim,npar)
        Integer, Intent (In)                 :: itt(6)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Local Scalars ..
        Integer                              :: k
!       .. Intrinsic Procedures ..
        Intrinsic                            :: modulo
!       .. 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) Then
            Write (nout,*)
            Write (nout,*) '* Current global optimum candidate:'
            Do k = 1, ndim
              Write (nout,99999) k, xb(k)
            End Do
            Write (nout,*) '* Current global optimum value:'
            Write (nout,99998) fb
          End If
        End If

!       If required set INFORM<0 to force exit
        Flush (nout)

        Return
99999   Format (1X,'* xb(',I3,') = ',F9.2)
99998   Format (1X,'* fb = ',F9.5)

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

!       .. Use Statements ..
        Use nag_library, Only: x04cbf
!       .. Parameters ..
        Integer, Parameter                   :: indent = 0, ncols = 80
        Character (10), Parameter            :: clabs(1:2) = (/               &
                                                'x_target  ','xb        ' /)
        Character (1), Parameter             :: diag = 'N', labcol = 'C',     &
                                                labrow = 'I', matrix = 'G'
        Character (4), Parameter             :: form = 'f9.2'
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: fb
        Integer, Intent (In)                 :: inform, ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: ruser(*), xb(ndim)
        Integer, Intent (In)                 :: itt(6)
!       .. Local Scalars ..
        Integer                              :: ifail, ldxcom
        Character (ncols)                    :: title
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable      :: xcom(:,:)
!       .. Executable Statements ..
!       Display final counters.
        Write (nout,99996) 'Total complete iterations            : ', itt(1)
        Write (nout,99996) 'Complete iterations since improvement : ', itt(2)
        Write (nout,99996) 'Total particles converged to xb       : ', itt(3)
        Write (nout,99996) 'Total improvements to global optimum  : ', itt(4)
        Write (nout,99996) 'Total function evaluations            : ', itt(5)
        Write (nout,99996) 'Total particles re-initialized        : ', itt(6)

!       Display why finalization occurred.
        Write (nout,*)
        Select Case (inform)
        Case (1)
          Write (nout,99995) 'Target value achieved'
        Case (2)
          Write (nout,99995) 'Minimum swarm standard deviation obtained'
        Case (3)
```

```
          Write (nout,99995) 'Sufficient particles converged'
        Case (4)
          Write (nout,99995) 'No improvement in preset iteration limit'
        Case (5)
          Write (nout,99995) 'Maximum complete iterations attained'
        Case (6)
          Write (nout,99995) 'Maximum function evaluations exceeded'
        Case (:-1)
          Write (nout,99994) inform
        Case Default
        End Select

!       Display final objective value and location.
        Write (nout,*)
        Write (nout,99999) 0.0E0_nag_wp
        Write (nout,99998) fb
        Flush (nout)

        ldxcom = ndim
        Allocate (xcom(ldxcom,2))
        xcom(1:ndim,1) = x_target(1:ndim)
        xcom(1:ndim,2) = xb(1:ndim)

        title = ' Comparison between known and achieved optima.'
        ifail = 0
        Call x04cbf(matrix,diag,ndim,2,xcom,ldxcom,form,title,labrow,clabs, &
          labcol,clabs,ncols,indent,ifail)

        Deallocate (xcom)

        Write (nout,99997)

        Return
99999   Format ('  Known objective optimum  : ',F13.5)
99998   Format ('  Achieved objective value : ',F13.5)
99997   Format ('**--------------------**--------------------**'/)
99996   Format (2X,A40,I16)
99995   Format (2X,A43)
99994   Format ('  User termination case                  :',I16)
      End Subroutine display_result
    End Module e05safe_mod

    Program e05safe
!     E05SAF Example Main Program

!     This example program demonstrates how to use E05SAF in standard
!       execution, and with a selection of coupled local minimizers.
!     The non-default option 'REPEATABILITY ON' is used here, giving
!       repeatable results.

!     .. Use Statements ..
      Use nag_library, Only: e05saf, e05zkf
      Use e05safe_mod, Only: display_result, iuser_reference, liopts, lopts,   &
                             monmod, nag_wp, ndim, nout, objfun_schwefel,      &
                             ruser_global
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: fb
      Integer                             :: ifail, inform, liuser, lruser,   &
                                             max_threads, npar
      Character (80)                      :: optstr
!     .. Local Arrays ..
      Real (Kind=nag_wp)                  :: bl(ndim), bu(ndim), opts(lopts), &
                                             xb(ndim)
      Real (Kind=nag_wp), Allocatable     :: ruser(:)
      Integer                             :: iopts(liopts), itt(6)
      Integer, Allocatable                :: iuser(:)
!     .. External Procedures ..
      Integer, External                   :: omp_get_max_threads
!     .. Intrinsic Procedures ..
```

```
      Intrinsic                          :: max
!     .. Executable Statements ..
!     Print advisory information.
      Write (nout,*) 'E05SAF Example Program Results'
      Write (nout,*)
      Write (nout,*) 'Minimization of the Schwefel function.'
      Write (nout,*)

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

!     Determine the number of threads to be used in the simulation
      max_threads = 1
      !$ max_threads = omp_get_max_threads()

!     Ensure NPAR is sufficient for the number of threads
      npar = max(4000,5*max_threads)

!     Assign sufficient sizes for iuser and ruser.
!     In this example, RUSER will be split into workspace unique to each thread,
!     and IUSER will be used to store the indexes to the first element of
!     workspace for each thread.

      liuser = max(30,iuser_reference+max_threads)
      lruser = ruser_global + 2*max_threads*ndim
      Allocate (iuser(liuser),ruser(lruser))

      Write (nout,*) '* Solution without using coupled local minimizer *'

!     Call E05ZKF to initialize the option arrays
      ifail = 0
      Call e05zkf('Initialize = E05SAF',iopts,liopts,opts,lopts,ifail)

!     Set the option SMP Callback Thread Safe to indicate the callback functions
!     are indeed threadsafe.  This must be done if using multiple threads.

      ifail = 0
      Call e05zkf('SMP Callback Thread Safe = Yes',iopts,liopts,opts,lopts, &
        ifail)

!     Set various options to non-default values if required.
      ifail = 0
      Call e05zkf('Repeatability = On',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Verify Gradients = Off',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Boundary = Hyperspherical',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Maximum iterations static = 150',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Call e05zkf('Maximum function evaluations = 50000000',iopts,liopts,opts, &
        lopts,ifail)
      ifail = 0
      Call e05zkf('Repulsion Initialize = 50',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Repulsion Finalize = 30',iopts,liopts,opts,lopts,ifail)

!     Set an objective target.
      ifail = 0
      Write (optstr,99999) 'Target Objective Value', 1.0E0_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Target Objective Tolerance', 1.0E-5_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Target Objective Safeguard', 1.0E-2_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
```

```
!     Set some additional SMP library options if required.
      ifail = 0
      Call e05zkf('SMP thread overrun = 100',iopts,liopts,opts,lopts,ifail)

      ifail = 0
      Write (optstr,99998) 'SMP Subswarm', max_threads
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)

!     Call E05SAF to search for the global optimum.
!     As guaranteed success cannot by default be achieved, IFAIL=1,-1 is
!     recommended on entry.
      ifail = -1
      Call e05saf(ndim,npar,xb,fb,bl,bu,objfun_schwefel,monmod,iopts,opts, &
        iuser,ruser,itt,inform,ifail)
!     It is essential to test IFAIL on exit.
      Select Case (ifail)
      Case (0,1)
!       E05SAF encountered no errors during operation,
!       and will have returned the best found optimum.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      Case (-999,-399,2:)
!       An error was detected.
        Continue
      Case Default
!       An instruction to exit was received by E05SAF from OBJFUN or MONMOD.
!       The exit flag will have been returned in INFORM.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      End Select

!     ------------------------------------------------------------------
      Write (nout,*) '* Solution using coupled local minimizer E04JYF *'



!     Set the local minimizer to be E04JYF and set corresponding options.
      ifail = 0
      Call e05zkf('Local Minimizer = E04JYF',iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Call e05zkf('Local Interior Iterations = 30',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Call e05zkf('Local Exterior Iterations = 200',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Write (optstr,99999) 'Local Interior Tolerance', 1.0E-4_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)
      ifail = 0
      Write (optstr,99999) 'Local Exterior Tolerance', 1.0E-8_nag_wp
      Call e05zkf(optstr,iopts,liopts,opts,lopts,ifail)

!     Call E05SAF to search for the global optimum.
      ifail = -1
      Call e05saf(ndim,npar,xb,fb,bl,bu,objfun_schwefel,monmod,iopts,opts, &
        iuser,ruser,itt,inform,ifail)

!     It is essential to test IFAIL on exit.
      Select Case (ifail)
      Case (0,1)
!       E05SAF encountered no errors during operation,
!       and will have returned the best found optimum.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      Case (-999,-399,2:)
!       An error was detected.
        Continue
      Case Default
!       An instruction to exit was received by E05SAF from OBJFUN or MONMOD.
!       The exit flag will have been returned in INFORM.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      End Select

!     ------------------------------------------------------------------
```

```
      Write (nout,*) '* Solution using coupled local minimizer E04UCF *'

!     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 Iterations = 10',iopts,liopts,opts,lopts, &
        ifail)
      ifail = 0
      Call e05zkf('Local Exterior Iterations = 100',iopts,liopts,opts,lopts, &
        ifail)

!     Call E05SAF to search for the global optimum.
      ifail = -1
      Call e05saf(ndim,npar,xb,fb,bl,bu,objfun_schwefel,monmod,iopts,opts, &
        iuser,ruser,itt,inform,ifail)

!     It is essential to test IFAIL on exit.
      Select Case (ifail)
      Case (0,1)
!       E05SAF encountered no errors during operation,
!       and will have returned the best found optimum.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      Case (-999,-399,2:)
!       An error was detected.
        Continue
      Case Default
!       An instruction to exit was received by E05SAF from OBJFUN or MONMOD.
!       The exit flag will have been returned in INFORM.
        Call display_result(ndim,xb,fb,itt,inform,ruser)
      End Select

99999 Format (A,' = ',E32.16)
99998 Format (A,' = ',I20)
    End Program e05safe
```

10.2 Program Data

None.

10.3 Program Results

```
 E05SAF Example Program Results

 Minimization of the Schwefel function.

 * Solution without using coupled local minimizer *
  Total complete iterations            :             2245
  Complete iterations since improvement :                1
  Total particles converged to xb      :                0
  Total improvements to global optimum  :             1573
  Total function evaluations           :          9882001
  Total particles re-initialized       :                0

                        Target value achieved

  Known objective optimum  :       0.00000
  Achieved objective value :       0.86400
  Comparison between known and achieved optima.
     x_target       xb
  1   -420.97  -420.54
  2   -420.97  -421.09
  3   -420.97  -420.70
  4   -420.97  -420.94
  5   -420.97  -421.14
  6   -420.97  -420.93
  7   -420.97  -420.59
  8   -420.97  -421.20
  9   -420.97  -421.04
 10   -420.97  -422.64
```

```
 11   -420.97  -420.43
 12   -420.97  -421.07
 13   -420.97  -421.31
 14   -420.97  -421.50
 15   -420.97  -422.12
 16   -420.97  -420.96
 17   -420.97  -421.22
 18   -420.97  -421.05
 19   -420.97  -421.92
 20   -420.97  -421.70
**--------------------**--------------------**

 * Solution using coupled local minimizer E04JYF *
  Total complete iterations             :             1075
  Complete iterations since improvement :                1
  Total particles converged to xb       :                0
  Total improvements to global optimum  :               50
  Total function evaluations            :          4742115
  Total particles re-initialized        :                0

                       Target value achieved

  Known objective optimum  :       0.00000
  Achieved objective value :      -0.00000
  Comparison between known and achieved optima.
     x_target       xb
 1   -420.97  -420.97
 2   -420.97  -420.97
 3   -420.97  -420.97
 4   -420.97  -420.97
 5   -420.97  -420.97
 6   -420.97  -420.97
 7   -420.97  -420.97
 8   -420.97  -420.97
 9   -420.97  -420.97
10   -420.97  -420.97
11   -420.97  -420.97
12   -420.97  -420.97
13   -420.97  -420.97
14   -420.97  -420.97
15   -420.97  -420.97
16   -420.97  -420.97
17   -420.97  -420.97
18   -420.97  -420.97
19   -420.97  -420.97
20   -420.97  -420.97
**--------------------**--------------------**

 * Solution using coupled local minimizer E04UCF *
  Total complete iterations             :             1387
  Complete iterations since improvement :                1
  Total particles converged to xb       :                0
  Total improvements to global optimum  :               64
  Total function evaluations            :          6107056
  Total particles re-initialized        :                0

                       Target value achieved

  Known objective optimum  :       0.00000
  Achieved objective value :       0.00000
  Comparison between known and achieved optima.
     x_target       xb
 1   -420.97  -420.97
 2   -420.97  -420.97
 3   -420.97  -420.97
 4   -420.97  -420.97
 5   -420.97  -420.97
 6   -420.97  -420.97
 7   -420.97  -420.97
 8   -420.97  -420.97
 9   -420.97  -420.97
```

```
 10   -420.97  -420.97
 11   -420.97  -420.97
 12   -420.97  -420.97
 13   -420.97  -420.97
 14   -420.97  -420.97
 15   -420.97  -420.97
 16   -420.97  -420.97
 17   -420.97  -420.97
 18   -420.97  -420.97
 19   -420.97  -420.97
 20   -420.97  -420.97
**--------------------**--------------------**
```

11 Algorithmic Details

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

INITIALIZE for $j = 1, n_p$
$\mathbf{x}_j = \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$
$\hat{\mathbf{x}}_j = \mathbf{R} \in U(\boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$
$\mathbf{v}_j = \mathbf{R} \in U(-\mathbf{V}_{\max}, \mathbf{V}_{\max})$
$\hat{f}_j = F(\hat{\mathbf{x}}_j)$
initialize w_j

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

end for
$\tilde{\mathbf{x}} = \frac{1}{2}(\boldsymbol{\ell}_{\text{box}} + \mathbf{u}_{\text{box}})$
$\tilde{f} = F(\tilde{\mathbf{x}})$
$I_c = I_s = 0$

SWARM while (not finalized),
$I_c = I_c + 1$
for $j = 1, n_p$
$\mathbf{x}_j = \text{BOUNDARY}(\mathbf{x}_j, \boldsymbol{\ell}_{\text{box}}, \mathbf{u}_{\text{box}})$
$f_j = F(\mathbf{x}_j)$
if $\left(f_j < \hat{f}_j\right)$
$\hat{f}_j = f_j$, $\hat{\mathbf{x}}_j = \mathbf{x}_j$
if $(f_j < \tilde{f})$
$\tilde{f} = f_j$, $\tilde{\mathbf{x}} = \mathbf{x}_j$
end for
if $(\text{new}(\tilde{f}))$
LOCMIN $(\tilde{\mathbf{x}}, \tilde{f}, O_i)$, $I_s = 0$
[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 $(\tilde{\mathbf{x}}, \tilde{f}, O_e)$

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

$\mathbf{v}_j$ velocity of particle j

w_j weight on $\mathbf{v}_j$ for velocity update, decreasing according to **Weight Decrease**

$V_{\max}$ maximum absolute velocity, dependent upon **Maximum Variable Velocity**

I_c swarm iteration counter

I_s iterations since $\tilde{\mathbf{x}}$ was updated

$\mathbf{D}_1$,$\mathbf{D}_2$ 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

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

O_i local optimizer interior options

O_e local optimizer exterior options

LOCMIN $(\mathbf{x}, f, O)$
apply local optimizer using the set of options O using the solution $(\mathbf{x}, f)$ 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, 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

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
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
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 ϵ 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 E05SAF 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 E05SAF 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. $\hat{\mathbf{x}}_j$ and $\hat{f}_j$ 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.

Distance Scaling a Default = ON

Determines whether distances should be scaled by box widths.

ON

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

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

Local Boundary Restriction r Default = 0.5

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

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

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 E05SAF.

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 × NDIM)	max(30, 3 × 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 E05SAF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SAF.

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 E05SAF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SAF.

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 the objective function that 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 E05SAF will have been acted upon by **Local Boundary Restriction**. As such, the domain exposed may be greatly smaller than that provided to E05SAF.

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 E05SAF will exit with IFAIL $= 1$ and INFORM $= 5$.

Unless set, this adapts to the parameters passed to E05SAF.

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 E05SAF 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, E05SAF 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: $r > 0.0$.

Optimize a Default $=$ MINIMIZE

Determines whether to maximize or minimize the objective function.

MINIMIZE
: The objective function will be minimized.

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

Repeatability a Default $=$ OFF

Allows for the same random number generator seed to be used for every call to E05SAF. **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: $i \geq 0$.

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, E05SAF 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 E05SAF, 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) E05SAF 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, E05SAF 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, E05SAF 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 = 100.0ϵ

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.

OBJECTIVE
FULL
: A more expensive gradient check will be performed on the gradients corresponding to the objective OBJFUN.

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 E05SAF 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 E05SAF 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.

An additional example program, `e05safe_smp.f90`, is included with the distribution material of multi-threaded implementations of the NAG Library to illustrate how to safely access independent subsections of the provided IUSER and RUSER arrays from multiple threads.

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 and FBEST 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.