NAG Library Routine Document

E05USF

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.

1 Purpose

E05USF is designed to find the global minimum of an arbitrary smooth sum of squares function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) by generating a number of different starting points and performing a local search from each using sequential quadratic programming.

2 Specification

```
SUBROUTINE E05USF (M, N, NCLIN, NCNLN, A, LDA, BL, BU, Y, CONFUN,         &
                   OBJFUN, NPTS, X, LDX, START, REPEAT1, NB, OBJF, F,      &
                   FJAC, LDFJAC, SDFJAC, ITER, C, LDC, CJAC, LDCJAC,       &
                   SDCJAC, CLAMDA, LDCLDA, ISTATE, LISTAT, IOPTS, OPTS,    &
                   IUSER, RUSER, INFO, IFAIL)

INTEGER            M, N, NCLIN, NCNLN, LDA, NPTS, LDX, NB, LDFJAC,         &
                   SDFJAC, ITER(NB), LDC, LDCJAC, SDCJAC, LDCLDA,          &
                   ISTATE(LISTAT,*), LISTAT, IOPTS(740), IUSER(*),         &
                   INFO(NB), IFAIL
REAL (KIND=nag_wp) A(LDA,*), BL(N+NCLIN+NCNLN), BU(N+NCLIN+NCNLN),         &
                   Y(M), X(LDX,*), OBJF(NB), F(M,*),                       &
                   FJAC(LDFJAC,SDFJAC,*), C(LDC,*),                        &
                   CJAC(LDCJAC,SDCJAC,*), CLAMDA(LDCLDA,*), OPTS(485),     &
                   RUSER(*)
LOGICAL            REPEAT1
EXTERNAL           CONFUN, OBJFUN, START
```

Before calling E05USF, the optional parameter arrays IOPTS and OPTS **must** be initialized for use with E05USF by calling E05ZKF with OPTSTR set to '`Initialize = e05usf`'. Optional parameters may subsequently be specified by calling E05ZKF before the call to E05USF.

3 Description

The local minimization method is E04USA. The problem is assumed to be stated in the following form:

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

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

E05USF solves a user-specified number of local optimization problems with different starting points. You may specify the starting points via the subroutine START. If a random number generator is used to generate the starting points then the argument REPEAT1 allows you to specify whether a repeatable set of points are generated or whether different starting points are generated on different calls. The resulting local minima are ordered and the best NB results returned in order of ascending values of the resulting objective function values at the minima. Thus the value returned in position 1 will be the best result obtained. If a sufficiently high number of different points are chosen then this is likely to be the global minimum.

4 References

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

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

5 Arguments

1: M – INTEGER *Input*

On entry: m, the number of subfunctions associated with $F(x)$.

Constraint: $\mathrm{M} > 0$.

2: N – INTEGER *Input*

On entry: n, the number of variables.

Constraint: $\mathrm{N} > 0$.

3: NCLIN – INTEGER *Input*

On entry: n_L, the number of general linear constraints.

Constraint: $\mathrm{NCLIN} \geq 0$.

4: NCNLN – INTEGER *Input*

On entry: n_N, the number of nonlinear constraints.

Constraint: $\mathrm{NCNLN} \geq 0$.

5: $\mathrm{A}(\mathrm{LDA},*)$ – REAL (KIND=nag_wp) array *Input*

Note: the second dimension of the array A must be at least N if $\mathrm{NCLIN} > 0$, and at least 1 otherwise.

On entry: the matrix A_L of general linear constraints in (1). That is, the ith row contains the coefficients of the ith general linear constraint, for $i = 1, 2, \ldots, \mathrm{NCLIN}$.

If $\mathrm{NCLIN} = 0$, the array A is not referenced.

6: LDA – INTEGER *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which E05USF is called.

Constraint: $\mathrm{LDA} \geq \mathrm{NCLIN}$.

7: $\mathrm{BL}(\mathrm{N} + \mathrm{NCLIN} + \mathrm{NCNLN})$ – REAL (KIND=nag_wp) array *Input*
8: $\mathrm{BU}(\mathrm{N} + \mathrm{NCLIN} + \mathrm{NCNLN})$ – REAL (KIND=nag_wp) array *Input*

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

Constraints:

$\mathrm{BL}(j) \leq \mathrm{BU}(j)$, for $j = 1, 2, \ldots, \mathrm{N} + \mathrm{NCLIN} + \mathrm{NCNLN}$;
if $\mathrm{BL}(j) = \mathrm{BU}(j) = \beta$, $|\beta| < bigbnd$.

9: Y(M) – REAL (KIND=nag_wp) array *Input*

On entry: the coefficients of the constant vector y of the objective function.

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

CONFUN must calculate the vector $c(x)$ of nonlinear constraint functions and (optionally) its Jacobian ($= \frac{\partial c}{\partial x}$) for a specified n-element vector x. If there are no nonlinear constraints (i.e., NCNLN $= 0$), CONFUN will never be called by E05USF and CONFUN may be the dummy routine E04UDM. (E04UDM is included in the NAG Library.) If there are nonlinear constraints, the first call to CONFUN will occur before the first call to OBJFUN.

The specification of CONFUN is:

```
SUBROUTINE CONFUN (MODE, NCNLN, N, LDCJSL, NEEDC, X, C, CJSL,         &
                   NSTATE, IUSER, RUSER)
INTEGER            MODE, NCNLN, N, LDCJSL, NEEDC(NCNLN), NSTATE,      &
                   IUSER(*)
REAL (KIND=nag_wp) X(N), C(NCNLN), CJSL(LDCJSL,*), 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 i such that NEEDC$(i) > 0$:

MODE $= 0$
: C(i), the ith nonlinear constraint.

MODE $= 1$
: All available elements in the ith row of CJSL.

MODE $= 2$
: C(i) and all available elements in the ith row of CJSL.

On exit: may be set to a negative value if you wish to abandon the solution to the current local minimization problem. In this case E05USF will move to the next local minimization problem.

2: NCNLN – INTEGER *Input*

On entry: n_N, the number of nonlinear constraints.

3: N – INTEGER *Input*

On entry: n, the number of variables.

4: LDCJSL – INTEGER *Input*

On entry: LDCJSL is the first dimension of the array CJSL.

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

On entry: the indices of the elements of C and/or CJSL that must be evaluated by CONFUN. If NEEDC$(i) > 0$, C(i) and/or the available elements of the ith row of CJSL (see argument MODE) must be evaluated at x.

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

On exit: if $\text{NEEDC}(i) > 0$ and $\text{MODE} = 0$ or 2, $\text{C}(i)$ must contain the value of $c_i(x)$. The remaining elements of C, corresponding to the non-positive elements of NEEDC, need not be set.

8: $\text{CJSL}(\text{LDCJSL}, *)$ – REAL (KIND=nag_wp) array *Input/Output*

CJSL may be regarded as a two-dimensional 'slice' of the three-dimensional array CJAC of E05USF.

On entry: unless **Derivative Level** $= 2$ or 3, the elements of CJSL are set to special values which enable E05USF to detect whether they are changed by CONFUN.

On exit: if $\text{NEEDC}(i) > 0$ and $\text{MODE} = 1$ or 2, the ith row of CJSL must contain the available elements of the vector ∇c_i given by

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

where $\frac{\partial c_i}{\partial x_j}$ is the partial derivative of the ith constraint with respect to the jth variable, evaluated at the point x. See also the argument NSTATE. The remaining rows of CJSL, corresponding to non-positive elements of NEEDC, need not be set.

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

Note that constant nonzero elements do affect the values of the constraints. Thus, if $\text{CJSL}(i, j)$ is set to a constant value, it need not be reset in subsequent calls to CONFUN, but the value $\text{CJSL}(i, j) \times \text{X}(j)$ must nonetheless be added to $\text{C}(i)$. For example, if $\text{CJSL}(1, 1) = 2$ and $\text{CJSL}(1, 2) = -5$ then the term $2 \times \text{X}(1) - 5 \times \text{X}(2)$ must be included in the definition of $\text{C}(1)$.

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

9: NSTATE – INTEGER *Input*

On entry: if $\text{NSTATE} = 1$ then E05USF is calling CONFUN for the first time on the current local optimization problem. This argument setting allows you to save computation time if certain data must be read or calculated only once.

10: $\text{IUSER}(*)$ – INTEGER array *User Workspace*

11: $\text{RUSER}(*)$ – REAL (KIND=nag_wp) array *User Workspace*

CONFUN is called with the arguments IUSER and RUSER as supplied to E05USF. You should use the arrays IUSER and RUSER to supply information to CONFUN.

CONFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E05USF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

CONFUN should be tested separately before being used in conjunction with E05USF. See also the description of the optional parameter **Verify**.

11: OBJFUN – SUBROUTINE, supplied by the user. *External Procedure*

OBJFUN must calculate either the ith element of the vector $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))^{\mathrm{T}}$ or all m elements of $f(x)$ and (optionally) its Jacobian ($= \dfrac{\partial f}{\partial x}$) for a specified n-element vector x.

The specification of OBJFUN is:

```
SUBROUTINE OBJFUN (MODE, M, N, LDFJSL, NEEDFI, X, F, FJSL,          &
                   NSTATE, IUSER, RUSER)

INTEGER            MODE, M, N, LDFJSL, NEEDFI, NSTATE, IUSER(*)
REAL (KIND=nag_wp) X(N), F(M), FJSL(LDFJSL,*), RUSER(*)
```

1: MODE – INTEGER *Input/Output*

On entry: indicates which values must be assigned during each call of OBJFUN. Only the following values need be assigned:

MODE $= 0$ and NEEDFI $= i$, where $i > 0$
F(i).

MODE $= 0$ and NEEDFI < 0
F.

MODE $= 1$ and NEEDFI < 0
All available elements of FJSL.

MODE $= 2$ and NEEDFI < 0
F and all available elements of FJSL.

On exit: may be set to a negative value if you wish to abandon the solution to the current local minimization problem. In this case E05USF will move to the next local minimization problem.

2: M – INTEGER *Input*

On entry: m, the number of subfunctions.

3: N – INTEGER *Input*

On entry: n, the number of variables.

4: LDFJSL – INTEGER *Input*

On entry: LDFJSL is the first dimension of the array FJSL.

5: NEEDFI – INTEGER *Input*

On entry: if NEEDFI $= i > 0$, only the ith element of $f(x)$ needs to be evaluated at x; the remaining elements need not be set. This can result in significant computational savings when $m \gg n$.

6: X(N) – REAL (KIND=nag_wp) array *Input*

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

7: F(M) – REAL (KIND=nag_wp) array *Output*

On exit: if MODE $= 0$ and NEEDFI $= i > 0$, F(i) must contain the value of f_i at x.

If MODE $= 0$ or 2 and NEEDFI < 0, F(i) must contain the value of f_i at x, for $i = 1, 2, \ldots, m$.

8: FJSL(LDFJSL, $*$) – REAL (KIND=nag_wp) array *Input/Output*

FJSL may be regarded as a two-dimensional 'slice' of the three-dimensional array FJAC of E05USF.

On entry: is set to a special value.

On exit: if MODE $= 1$ or 2 and NEEDFI < 0, the ith row of FJSL must contain the available elements of the vector ∇f_i given by

$$\nabla f_i = (\partial f_i / \partial x_1, \partial f_i / \partial x_2, \ldots, \partial f_i / \partial x_n)^{\mathrm{T}},$$

evaluated at the point x. See also the argument NSTATE.

9: NSTATE – INTEGER *Input*

On entry: if NSTATE $= 1$ then E05USF is calling OBJFUN for the first time on the current local optimization problem. This argument setting allows you to save computation time if certain data must be read or calculated only once.

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

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

OBJFUN is called with the arguments IUSER and RUSER as supplied to E05USF. You should use the arrays IUSER and RUSER to supply information to OBJFUN.

OBJFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which E05USF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

OBJFUN should be tested separately before being used in conjunction with E05USF. See also the description of the optional parameter **Verify**.

12: NPTS – INTEGER *Input*

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

Constraint: $1 \leq$ NB $\leq$ NPTS.

13: X(LDX, $*$) – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array X must be at least NB.

On exit: X(j, i) contains the final estimate of the ith solution, for $j = 1, 2, \ldots,$ N.

14: LDX – INTEGER *Input*

On entry: the first dimension of the array X as declared in the (sub)program from which E05USF is called.

Constraint: LDX $\geq$ N.

15: START – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

START must calculate the NPTS starting points to be used by the local optimizer. If you do not wish to write a routine specific to your problem then E05UCZ may be used as the actual argument. E05UCZ is supplied in the NAG Library and uses the NAG quasi-random number generators to distribute starting points uniformly across the domain. It is affected by the value of REPEAT1.

The specification of START is:

```
SUBROUTINE START (NPTS, QUAS, N, REPEAT1, BL, BU, IUSER, RUSER,      &
                  MODE)
INTEGER            NPTS, N, IUSER(*), MODE
REAL (KIND=nag_wp) QUAS(N,NPTS), BL(N), BU(N), RUSER(*)
LOGICAL            REPEAT1
```

1: NPTS – INTEGER *Input*

On entry: indicates the number of starting points.

2: QUAS(N, NPTS) – REAL (KIND=nag_wp) array *Input/Output*

On entry: all elements of QUAS will have been set to zero, so only nonzero values need be set subsequently.

On exit: must contain the starting points for the NPTS local minimizations, i.e., QUAS(j, i) must contain the jth component of the ith starting point.

3: N – INTEGER *Input*

On entry: the number of variables.

4: REPEAT1 – LOGICAL *Input*

On entry: specifies whether a repeatable or non-repeatable sequence of points are to be generated.

5: BL(N) – REAL (KIND=nag_wp) array *Input*

On entry: the lower bounds on the variables. These may be used to ensure that the starting points generated in some sense 'cover' the region, but there is no requirement that a starting point be feasible.

6: BU(N) – REAL (KIND=nag_wp) array *Input*

On entry: the upper bounds on the variables. (See BL.)

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

START is called with the arguments IUSER and RUSER as supplied to E05USF. You should use the arrays IUSER and RUSER to supply information to START.

9: MODE – INTEGER *Input/Output*

On entry: MODE will contain 0.

On exit: if you set MODE to a negative value then E05USF will terminate immediately with IFAIL $= 9$.

START must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which E05USF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

16: REPEAT1 – LOGICAL *Input*

On entry: is passed as an argument to START and may be used to initialize a random number generator to a repeatable, or non-repeatable, sequence. See Section 9 for more detail.

17: NB – INTEGER *Input*

On entry: the number of solutions to be returned. The routine saves up to NB local minima ordered by increasing value of the final objective function. If the defining criterion for 'best solution' is only that the value of the objective function is as small as possible then NB should be set to 1. However, if you want to look at other solutions that may have desirable properties then setting $\text{NB} > 1$ will produce NB local minima, ordered by increasing value of their objective functions at the minima.

Constraint: $1 \leq \text{NB} \leq \text{NPTS}$.

18: OBJF(NB) – REAL (KIND=nag_wp) array *Output*

On exit: $\text{OBJF}(i)$ contains the value of the objective function at the final iterate for the ith solution.

19: $\text{F}(\text{M}, *)$ – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array F must be at least NB.

On exit: $\text{F}(j, i)$ contains the value of the jth function f_j at the final iterate, for $j = 1, 2, \ldots, \text{M}$, for the ith solution, for $i = 1, 2, \ldots, \text{NB}$.

20: $\text{FJAC}(\text{LDFJAC}, \text{SDFJAC}, *)$ – REAL (KIND=nag_wp) array *Output*

Note: the last dimension of the array FJAC must be at least NB.

On exit: for the ith returned solution, the Jacobian matrix of the functions $f_1, f_2, \ldots, f_m$ at the final iterate, i.e., $\text{FJAC}(k, j, i)$ contains the partial derivative of the kth function with respect to the jth variable, for $k = 1, 2, \ldots, \text{M}$, $j = 1, 2, \ldots, \text{N}$ and $i = 1, 2, \ldots, \text{NB}$. (See also the discussion of argument FJSL under OBJFUN.)

21: LDFJAC – INTEGER *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which E05USF is called.

Constraint: $\text{LDFJAC} \geq \text{M}$.

22: SDFJAC – INTEGER *Input*

On entry: the second dimension of the array FJAC as declared in the (sub)program from which E05USF is called.

Constraint: $\text{SDFJAC} \geq \text{N}$.

23: ITER(NB) – INTEGER array *Output*

On exit: $\text{ITER}(i)$ contains the number of major iterations performed to obtain the ith solution. If less than NB solutions are returned then ITER(NB) contains the number of starting points that have resulted in a converged solution. If this is close to NPTS then this might be indicative that fewer than NB local minima exist.

24: $\text{C}(\text{LDC}, *)$ – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array C must be at least NB.

On exit: if $\text{NCNLN} > 0$, $\text{C}(j, i)$ contains the value of the jth nonlinear constraint function c_j at the final iterate, for the ith solution, for $j = 1, 2, \ldots, \text{NCNLN}$.

If $\text{NCNLN} = 0$, the array C is not referenced.

25: LDC – INTEGER *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which E05USF is called.

Constraint: LDC $\geq$ NCNLN.

26: CJAC(LDCJAC, SDCJAC, $*$) – REAL (KIND=nag_wp) array *Output*

Note: the last dimension of the array CJAC must be at least NB.

On exit: if NCNLN > 0, CJAC contains the Jacobian matrices of the nonlinear constraint functions at the final iterate for each of the returned solutions, i.e., CJAC(k, j, i) contains the partial derivative of the kth constraint function with respect to the jth variable, for $k = 1, 2, \ldots,$ NCNLN and $j = 1, 2, \ldots,$ N, for the ith solution. (See the discussion of argument CJSL under CONFUN.)

If NCNLN $= 0$, the array CJAC is not referenced.

27: LDCJAC – INTEGER *Input*

On entry: the first dimension of the array CJAC as declared in the (sub)program from which E05USF is called.

Constraint: LDCJAC $\geq$ NCNLN.

28: SDCJAC – INTEGER *Input*

On entry: the second dimension of the array CJAC as declared in the (sub)program from which E05USF is called.

Constraint: if NCNLN > 0, SDCJAC $\geq$ N.

29: CLAMDA(LDCLDA, $*$) – REAL (KIND=nag_wp) array *Output*

Note: the second dimension of the array CLAMDA must be at least NB.

On exit: the values of the QP multipliers from the last QP subproblem solved for the ith solution. CLAMDA(j, i) should be non-negative if ISTATE$(j, i) = 1$ and non-positive if ISTATE$(j, i) = 2$.

30: LDCLDA – INTEGER *Input*

On entry: the first dimension of the array CLAMDA as declared in the (sub)program from which E05USF is called.

Constraint: LDCLDA $\geq$ N + NCLIN + NCNLN.

31: ISTATE(LISTAT, $*$) – INTEGER array *Output*

Note: the second dimension of the array ISTATE must be at least NB.

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

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

32: LISTAT – INTEGER *Input*

On entry: the first dimension of the array ISTATE as declared in the (sub)program from which E05USF is called.

Constraint: LISTAT $\geq$ N + NCLIN + NCNLN.

33: IOPTS(740) – INTEGER array *Communication Array*

34: OPTS(485) – REAL (KIND=nag_wp) array *Communication Array*

The arrays IOPTS and OPTS **must not** be altered between calls to any of the routines E05USF and E05ZKF.

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

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

IUSER and RUSER are not used by E05USF, but are passed directly to CONFUN, OBJFUN and START and should be used to pass information to these routines.

With SMP-enabled versions of E05USF the arrays IUSER and RUSER provided are classified as OpenMP shared memory. Use of IUSER and RUSER has to take account of this in order to preserve thread safety whenever information is written back to either of these arrays.

37: INFO(NB) – INTEGER array *Output*

On exit: if IFAIL = 0, INFO(i) does not contain an error value returned by E04USF/E04USA .

If IFAIL = 8 on exit, then not all NB solutions have been found, and INFO(NB) contains the number of solutions actually found.

38: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output arguments may be useful even if IFAIL $\neq$ 0 on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL = 0 unless the routine detects an error or a warning has been flagged (see Section 6).

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).

Note: E05USF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL = 1

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

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

On entry, LDA = $\langle value \rangle$ and NCLIN = $\langle value \rangle$.
Constraint: LDA $\geq$ NCLIN.

On entry, LDC = $\langle value \rangle$ and NCNLN = $\langle value \rangle$.
Constraint: LDC $\geq$ NCNLN.

On entry, LDCJAC = ⟨*value*⟩ and NCNLN = ⟨*value*⟩.
Constraint: LDCJAC $\geq$ NCNLN.

On entry, LDCLDA = ⟨*value*⟩, N = ⟨*value*⟩, NCLIN = ⟨*value*⟩ and NCNLN = ⟨*value*⟩.
Constraint: LDCLDA $\geq$ N + NCLIN + NCNLN.

On entry, LDFJAC = ⟨*value*⟩ and M = ⟨*value*⟩.
Constraint: LDFJAC $\geq$ M.

On entry, LDX = ⟨*value*⟩ and N = ⟨*value*⟩.
Constraint: LDX $\geq$ N.

On entry, LISTAT = ⟨*value*⟩, N = ⟨*value*⟩, NCLIN = ⟨*value*⟩ and NCNLN = ⟨*value*⟩.
Constraint: LISTAT $\geq$ N + NCLIN + NCNLN.

On entry, M = ⟨*value*⟩.
Constraint: M $>$ 0.

On entry, N = ⟨*value*⟩.
Constraint: N $>$ 0.

On entry, NB = ⟨*value*⟩ and NPTS = ⟨*value*⟩.
Constraint: 1 $\leq$ NB $\leq$ NPTS.

On entry, NCLIN = ⟨*value*⟩.
Constraint: NCLIN $\geq$ 0.

On entry, NCNLN = ⟨*value*⟩.
Constraint: NCNLN $\geq$ 0.

On entry, NCNLN $>$ 0, SDCJAC = ⟨*value*⟩ and N = ⟨*value*⟩.
Constraint: if NCNLN $>$ 0, SDCJAC $\geq$ N.

On entry, SDFJAC = ⟨*value*⟩ and N = ⟨*value*⟩.
Constraint: SDFJAC $\geq$ N.

IFAIL = 2

*E05USF has terminated without finding any solutions. The majority of calls to the local optimizer have failed to find a feasible point for the linear constraints and bounds, which means that either no feasible point exists for the given value of the optional parameter **Linear Feasibility Tolerance** (default value $\sqrt{macheps}$, where macheps is the **machine precision**), or no feasible point could be found in the number of iterations specified by the optional parameter **Minor Iteration Limit**. You should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ, you should ensure that the value of the optional parameter **Linear Feasibility Tolerance** is greater than σ. For example, if all elements of A_L are of order unity and are accurate to only three decimal places, **Linear Feasibility Tolerance** should be at least 10^{-3}.*

No solution obtained. Linear constraints may be infeasible.

IFAIL = 3

E05USF has failed to find any solutions. The majority of local optimizations could not find a feasible point for the nonlinear constraints. The problem may have no feasible solution. This behaviour will occur if there is no feasible point for the nonlinear constraints. (However, there is no general test that can determine whether a feasible point exists for a set of nonlinear constraints.)

No solution obtained. Nonlinear constraints may be infeasible.

IFAIL = 4

E05USF has failed to find any solutions. The majority of local optimizations have failed because the limiting number of iterations have been reached.

No solution obtained. Many potential solutions reach iteration limit.

IFAIL $= 7$

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

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

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

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

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

IFAIL $= 8$

Only ⟨*value*⟩ solutions obtained.

Not all NB solutions have been found. INFO(NB) contains the number actually found.

IFAIL $= 9$

User terminated computation from START procedure: MODE = ⟨*value*⟩.

If E05UCZ has been used as an actual argument for START then the message displayed, when IFAIL $= 0$ *or* -1 *on entry to E05USF, will have the following meaning:*

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

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

IFAIL $= 10$

Failed to initialize optional parameter arrays.

IFAIL $= -99$

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

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

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

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

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

8 Parallelism and Performance

E05USF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library. 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. The user workspace arrays IUSER and RUSER are classified as OpenMP shared memory and use of IUSER and RUSER has to take account of this in order to preserve thread safety whenever information is written back to either of these arrays. If at all possible, it is recommended that these arrays are only used to supply read-only data to the user functions when a multithreaded implementation is being used.

E05USF 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.

9 Further Comments

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

The auxiliary routine E05UCZ makes use of the NAG quasi-random Sobol generator (G05YLF and G05YMF). If E05UCZ is used as the actual argument for START and REPEAT1 = .FALSE. then a randomly chosen value for ISKIP is used, otherwise ISKIP is set to 100. If REPEAT1 is set to .FALSE. and the program is executed several times, each time producing the same best answer, then there is increased probability that this answer is a global minimum. However, if it is important that identical results be obtained on successive runs, then REPEAT1 should be set to .TRUE..

9.1 Description of the Printed Output

See Section 9.1 in E04USF/E04USA.

10 Example

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

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

where

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

and

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

subject to the bounds

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

to the general linear constraint

$$x_1 + x_2 \geq 1.0$$

and to the nonlinear constraint

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

The optimal solution (to five figures) is

$$x^* = (0.41995, 1.28484)^{\mathrm{T}},$$

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

The document for E04UQF/E04UQA includes an example program to solve the same problem using some of the optional parameters described in Section 12.

10.1 Program Text

```
!   E05USF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module e05usfe_mod

!     E05USF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: confun, objfun, start
    Contains
      Subroutine objfun(mode,m,n,ldfjsl,needfi,x,f,fjsl,nstate,iuser,ruser)
!       Evaluates the subfunctions and their 1st derivatives.

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: a(44) = (/8._nag_wp,8._nag_wp,       &
                                          10._nag_wp,10._nag_wp,10._nag_wp,    &
                                          10._nag_wp,12._nag_wp,12._nag_wp,    &
                                          12._nag_wp,12._nag_wp,14._nag_wp,    &
```

```
                                              14._nag_wp,14._nag_wp,16._nag_wp,    &
                                              16._nag_wp,16._nag_wp,18._nag_wp,    &
                                              18._nag_wp,20._nag_wp,20._nag_wp,    &
                                              20._nag_wp,22._nag_wp,22._nag_wp,    &
                                              22._nag_wp,24._nag_wp,24._nag_wp,    &
                                              24._nag_wp,26._nag_wp,26._nag_wp,    &
                                              26._nag_wp,28._nag_wp,28._nag_wp,    &
                                              30._nag_wp,30._nag_wp,30._nag_wp,    &
                                              32._nag_wp,32._nag_wp,34._nag_wp,    &
                                              36._nag_wp,36._nag_wp,38._nag_wp,    &
                                              38._nag_wp,40._nag_wp,42._nag_wp/)
!       .. Scalar Arguments ..
        Integer, Intent (In)           :: ldfjsl, m, n, needfi, nstate
        Integer, Intent (Inout)        :: mode
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: f(m)
        Real (Kind=nag_wp), Intent (Inout) :: fjsl(ldfjsl,*), ruser(*)
        Real (Kind=nag_wp), Intent (In) :: x(n)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: temp
        Integer                        :: i
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..

!       This is a two-dimensional objective function.
!       As an example of using the mode mechanism,
!       terminate if any other problem size is supplied.

        If (n/=2) Then
          mode = -1
        End If

        If (nstate==1) Then
!         This is the first call.
!         Take any special action here if desired.
          Continue
        End If

        If (mode==0 .And. needfi>0) Then
          f(needfi) = x(1) + (0.49_nag_wp-x(1))*exp(-x(2)*(a(needfi)-          &
            8.0_nag_wp))
        Else
          Do i = 1, m
            temp = exp(-x(2)*(a(i)-8._nag_wp))

            If (mode==0 .Or. mode==2) Then
              f(i) = x(1) + (0.49_nag_wp-x(1))*temp
            End If

            If (mode==1 .Or. mode==2) Then
              fjsl(i,1) = 1._nag_wp - temp
              fjsl(i,2) = -(0.49_nag_wp-x(1))*(a(i)-8._nag_wp)*temp
            End If

          End Do
        End If

        Return
      End Subroutine objfun
      Subroutine confun(mode,ncnln,n,ldcjsl,needc,x,c,cjsl,nstate,iuser,ruser)
!       Evaluates the nonlinear constraints and their 1st derivatives.

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: ldcjsl, n, ncnln, nstate
        Integer, Intent (Inout)        :: mode
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: c(ncnln)
        Real (Kind=nag_wp), Intent (Inout) :: cjsl(ldcjsl,*), ruser(*)
        Real (Kind=nag_wp), Intent (In) :: x(n)
```

```
        Integer, Intent (Inout)        :: iuser(*)
        Integer, Intent (In)           :: needc(ncnln)
!       .. Executable Statements ..

!       This problem has only one constraint.
!       As an example of using the mode mechanism,
!       terminate if any other size is supplied.

        If (ncnln/=1) Then
          mode = -1
        End If

        If (nstate==1) Then

!         First call to CONFUN.  Set all Jacobian elements to zero.
!         Note that this will only work when 'Derivative Level = 3'
!         (the default; see Section 11.1 of the E04USA document).

          cjsl(1:ncnln,1:n) = 0._nag_wp
        End If

        If (needc(1)>0) Then

          If (mode==0 .Or. mode==2) Then
            c(1) = -0.09_nag_wp - x(1)*x(2) + 0.49_nag_wp*x(2)
          End If

          If (mode==1 .Or. mode==2) Then
            cjsl(1,1) = -x(2)
            cjsl(1,2) = -x(1) + 0.49_nag_wp
          End If

        End If

        Return
      End Subroutine confun
      Subroutine start(npts,quas,n,repeat1,bl,bu,iuser,ruser,mode)

!       Sets the initial points.
!       A typical user-defined start procedure.

!       .. Use Statements ..
        Use nag_library, Only: g05kgf, g05saf
!       .. Scalar Arguments ..
        Integer, Intent (Inout)        :: mode
        Integer, Intent (In)           :: n, npts
        Logical, Intent (In)           :: repeat1
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: bl(n), bu(n)
        Real (Kind=nag_wp), Intent (Inout) :: quas(n,npts), ruser(*)
        Integer, Intent (Inout)        :: iuser(*)
!       .. Local Scalars ..
        Integer                        :: genid, i, ifail, lstate, subid
!       .. Local Arrays ..
        Integer, Allocatable           :: state(:)
!       .. Executable Statements ..
!       quas is pre-assigned to zero.
        If (repeat1) Then
          quas(1,1) = 0.4_nag_wp
          quas(2,2) = 1._nag_wp
        Else
!         Generate a non-repeatable spread of points between bl and bu.
          genid = 2
          subid = 53
          lstate = -1
          Allocate (state(lstate))
          ifail = 0
          Call g05kgf(genid,subid,state,lstate,ifail)
          Deallocate (state)
          Allocate (state(lstate))
          ifail = 0
```

```
          Call g05kgf(genid,subid,state,lstate,ifail)
          Do i = 1, npts
            ifail = 0
            Call g05saf(n,state,quas(1,i),ifail)
            quas(1:n,i) = bl(1:n) + (bu(1:n)-bl(1:n))*quas(1:n,i)
          End Do
          Deallocate (state)
        End If
!       Set mode negative to terminate execution for any reason.
        mode = 0
        Return
      End Subroutine start
    End Module e05usfe_mod
    Program e05usfe

!     E05USF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: dgemv, e05usf, e05zkf, nag_wp
      Use e05usfe_mod, Only: confun, objfun, start
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: liopts = 740, lopts = 485, m = 44,   &
                                          n = 2, nb = 1, nclin = 1, ncnln = 1, &
                                          nin = 5, nout = 6, npts = 3
      Integer, Parameter               :: sdfjac = n
      Integer, Parameter               :: lda = nclin
      Integer, Parameter               :: ldc = ncnln
      Integer, Parameter               :: ldcjac = ncnln
      Integer, Parameter               :: ldclda = n + nclin + ncnln
      Integer, Parameter               :: ldfjac = m
      Integer, Parameter               :: ldx = n
      Integer, Parameter               :: listat = n + nclin + ncnln
      Logical, Parameter               :: repeat1 = .True.
!     .. Local Scalars ..
      Integer                          :: i, ifail, j, k, l, sda, sdcjac
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:,:), bl(:), bu(:), c(:,:),        &
                                          cjac(:,:,:), clamda(:,:), f(:,:),    &
                                          fjac(:,:,:), work(:), x(:,:), y(:)
      Real (Kind=nag_wp)               :: objf(nb), opts(lopts), ruser(1)
      Integer                          :: info(nb), iopts(liopts), iter(nb),   &
                                          iuser(1)
      Integer, Allocatable             :: istate(:,:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: max
!     .. Executable Statements ..
      Write (nout,*) 'E05USF Example Program Results'
      Flush (nout)

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

      If (nclin>0) Then
        sda = n
      Else
        sda = 1
      End If

      If (ncnln>0) Then
        sdcjac = n
      Else
        sdcjac = 0
      End If

      Allocate (a(lda,sda),bl(n+nclin+ncnln),bu(n+nclin+ncnln),y(m),c(ldc,nb), &
        cjac(ldcjac,sdcjac,nb),f(m,nb),fjac(ldfjac,sdfjac,nb),                 &
        clamda(ldclda,nb),istate(listat,nb),x(ldx,nb),work(max(1,nclin)))

      If (nclin>0) Then
```

```
          Read (nin,*)(a(i,1:sda),i=1,nclin)
        End If

        Read (nin,*) y(1:m)
        Read (nin,*) bl(1:(n+nclin+ncnln))
        Read (nin,*) bu(1:(n+nclin+ncnln))

!       Initialize the solver.

        ifail = 0
        Call e05zkf('Initialize = E05USF',iopts,liopts,opts,lopts,ifail)

!       Solve the problem.

        ifail = -1
        Call e05usf(m,n,nclin,ncnln,a,lda,bl,bu,y,confun,objfun,npts,x,ldx,      &
          start,repeat1,nb,objf,f,fjac,ldfjac,sdfjac,iter,c,ldc,cjac,ldcjac,     &
          sdcjac,clamda,ldclda,istate,listat,iopts,opts,iuser,ruser,info,ifail)

        Select Case (ifail)
        Case (0)
          l = nb
        Case (8)
          l = info(nb)
          Write (nout,99999) iter(nb)
        Case Default
          Go To 100
        End Select

loop:   Do i = 1, l
          Write (nout,99998) i
          Write (nout,99997) info(i)
          Write (nout,99996) 'Varbl'
          Do j = 1, n
            Write (nout,99995) 'V', j, istate(j,i), x(j,i), clamda(j,i)
          End Do
          If (nclin>0) Then
            Write (nout,99996) 'L Con'

!           Below is a call to the level 2 BLAS routine DGEMV.
!           This performs the matrix vector multiplication A*X
!           (linear constraint values) and puts the result in
!           the first NCLIN locations of WORK.

            Call dgemv('N',nclin,n,1.0_nag_wp,a,lda,x(1,i),1,0.0_nag_wp,work,1)

            Do k = n + 1, n + nclin
              j = k - n
              Write (nout,99995) 'L', j, istate(k,i), work(j), clamda(k,i)
            End Do
          End If
          If (ncnln>0) Then
            Write (nout,99996) 'N Con'
            Do k = n + nclin + 1, n + nclin + ncnln
              j = k - n - nclin
              Write (nout,99995) 'N', j, istate(k,i), c(j,i), clamda(k,i)
            End Do
          End If
          Write (nout,99994) objf(i)
          Write (nout,99993)
          Write (nout,99992)(clamda(k,i),k=1,n+nclin+ncnln)

          If (l==1) Then
            Exit loop
          End If

          Write (nout,*)

          Write (nout,*)                                                         &
            ' ------------------------------------------------------ '
```

```
      End Do loop

100   Continue

99999 Format (1X,I20,'starting points converged')
99998 Format (/,1X,'Solution number',I16)
99997 Format (/,1X,'Local minimization exited with code',I5)
99996 Format (/,1X,A,2X,'Istate',3X,'Value',9X,'Lagr Mult',/)
99995 Format (1X,A,2(1X,I3),4X,F12.4,2X,F12.4)
99994 Format (/,1X,'Final objective value = ',1X,F12.4)
99993 Format (/,1X,'QP multipliers')
99992 Format (1X,F12.4)
    End Program e05usfe
```

10.2 Program Data

```
E05USF Example Program Data
  1.0   1.0                          :End of matrix A
  0.49  0.49  0.48  0.47  0.48  0.47  0.46  0.46  0.45  0.43  0.45
  0.43  0.43  0.44  0.43  0.43  0.46  0.45  0.42  0.42  0.43  0.41
  0.41  0.40  0.42  0.40  0.40  0.41  0.40  0.41  0.41  0.40  0.40
  0.40  0.38  0.41  0.40  0.40  0.41  0.38  0.40  0.40  0.39  0.39 :End of Y
  0.4      -4.0      1.0      0.0      :End of BL
  1.0E+25   1.0E+25  1.0E+25  1.0E+25  :End of BU
```

10.3 Program Results

```
 E05USF Example Program Results

 Solution number               1

 Local minimization exited with code    0

 Varbl  Istate   Value         Lagr Mult

 V   1   0          0.4200        0.0000
 V   2   0          1.2848        0.0000

 L Con  Istate   Value         Lagr Mult

 L   1   0          1.7048        0.0000

 N Con  Istate   Value         Lagr Mult

 N   1   1        -0.0000        0.0334

 Final objective value =        0.0142

 QP multipliers
       0.0000
       0.0000
       0.0000
       0.0334
```

11 Algorithmic Details

See Section 11 in E04USF/E04USA.

12 Optional Parameters

Several optional parameters in E05USF define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of E05USF these optional parameters have associated *default values* that are appropriate for most problems. Therefore you need only specify those optional parameters whose values are to be different from their default values.

Optional parameters may be specified by calling E05ZKF before a call to E05USF. Before calling E05USF, the optional parameter arrays IOPTS and OPTS **must** be initialized for use with E05USF by calling E05ZKF with OPTSTR set to '`Initialize = e05usf`'.

All optional parameters not specified are set to their default values. Optional parameters specified are unaltered by E05USF (unless they define invalid values) and so remain in effect for subsequent calls to E05USF.

See Section 12 in E04USF/E04USA for full details.

The **Warm Start** option of E04USF/E04USA is not a valid option for use with E05USF.

E05USF supports two options that are distinct from those of E04USF/E04USA:

Punch Unit i Default $= 6$

This option allows you to send information arising from an appropriate setting of **Out_Level** to be sent to the Fortran unit number defined by **Punch Unit**. If you wish this file to be different to the standard output unit (6) where other output is displayed then this file should be attached by calling X04ACF prior to calling E05USF.

Out_Level i Default $= 0$

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

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

13 Description of Monitoring Information

See Section 13 in E04USF/E04USA.