NAG Fortran Library Routine Document

E05JBF

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 9 of this document. If, however, you wish to reset some or all of the settings please refer to Section 10 for a detailed description of the algorithm, and to Section 11 for a detailed description of the specification of the optional parameters*.

1 Purpose

E05JBF is designed to find the global minimum or maximum of an arbitrary function, subject to simple bound-constraints using a multilevel coordinate search method. Derivatives are not required, but convergence is only guaranteed if the objective function is continuous in a neighbourhood of a global optimum. It is not intended for large problems.

E05JBF uses **forward communication** for evaluating the objective function.

The initialization routine E05JAF **must** have been called before calling E05JBF.

2 Specification

```
 SUBROUTINE E05JBF (N, OBJFUN, MONIT, IBOUND, IINIT, BL, BU, NINIT,
1                   INLIST, NUMPTS, INITPT, LDIN, X, OBJ, RW, LENRW,
2                   IUSER, RUSER, IFAIL)

 INTEGER            N, IBOUND, IINIT, NINIT, NUMPTS(N), INITPT(N), LDIN,
1                   LENRW, IUSER(*), IFAIL
 double precision   BL(N), BU(N), INLIST(LDIN,NINIT), X(N), OBJ,
1                   RW(LENRW), RUSER(*)
 EXTERNAL           OBJFUN, MONIT
```

Before calling E05JBF, or any of the option-setting or option-getting routines E05JCF, E05JDF, E05JEF, E05JFF, E05JGF, E05JHF, E05JJF, E05JKF or E05JLF, E05JAF **must** be called. The specification for E05JAF is:

```
SUBROUTINE E05JAF (N, RW, LENRW, IFAIL)

INTEGER            N, LENRW, IFAIL
double precision   RW(LENRW)
```

E05JAF **must** be called with LENRW, the declared length of RW, satisfying:

$$\text{LENRW} \geq 100.$$

The contents of the array RW and the value of the parameter N **must not** be altered between calls of the routines E05JAF, E05JBF, E05JCF, E05JDF, E05JEF, E05JFF, E05JGF, E05JHF, E05JJF, E05JKF or E05JLF.

3 Description

E05JBF is designed to solve modestly-sized global optimization problems having simple bound-constraints only; it finds the global optimum of a nonlinear function subject to a set of bound constraints on the variables. Without loss of generality, the problem is assumed to be stated in the following form:

$$\underset{\mathbf{x}\in R^n}{\text{minimize}}\, F(\mathbf{x}) \quad \text{subject to} \quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \quad \text{and} \quad \mathbf{l} < \mathbf{u},$$

where $F(\mathbf{x})$ (the *objective function*) is a nonlinear scalar function (assumed to be continuous in a neighbourhood of a global minimum), and the bound vectors are elements of $\bar{R}^n$, where $\bar{R}$ denotes the extended reals $R \cup \{-\infty, \infty\}$. Relational operators between vectors are interpreted elementwise.

The optional parameter **Maximize** should be set if you wish to solve maximization, rather than minimization, problems.

If certain bounds are not present, the associated elements of **l** or **u** can be set to special values that will be treated as $-\infty$ or $+\infty$. See the description of the optional parameter **Infinite Bound Size**. Phrases in this document containing terms like 'unbounded values' should be understood to be taken relative to this optional parameter.

A typical excerpt from a routine calling E05JBF is:

```
CALL E05JAF (N, RW, LENRW, ...)
CALL E05JCF (IOPTS, RW, LENRW, ...)
CALL E05JBF (N, OBJFUN, ...)
```

where E05JCF reads a set of optional parameter definitions from the file with unit number IOPTS.

The initialization routine E05JAF does not need to be called before each invocation of E05JBF, so long as the problem size, N, remains unchanged. You should be aware that a call to the initialization routine will reset each optional parameter to its default value, and, if you are using repeatable randomized initialization lists (see the description of the parameter IINIT), the random seed stored in the array RW will be destroyed.

You must supply a (sub)program that defines $F(\mathbf{x})$; derivatives are not required.

The method used by E05JBF is based on MCS, the Multilevel Coordinate Search method described in Huyer and Neumaier (1999), and the algorithm it uses is described in detail in Section 10.

4 References

Huyer W and Neumaier A (1999) Global Optimization by Multilevel Coordinate Search *Journal of Global Optimization* **14** 331–355

5 Parameters

1: N – INTEGER *Input*

On entry: n, the number of variables.

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

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

OBJFUN must calculate the objective function $F(\mathbf{x})$ for a specified n-vector **x**.

Its specification is:

```
SUBROUTINE OBJFUN (N, X, F, NSTATE, IUSER, RUSER, INFORM)
INTEGER            N, NSTATE, IUSER(*), INFORM
double precision   X(N), F, RUSER(*)
```

1: N – INTEGER *Input*

On entry: n, the number of variables, as defined in the call to E05JBF.

2: X(N) – ***double precision*** array *Input*

On entry: **x**, the vector at which the objective function is to be evaluated.

3: F – ***double precision*** *Output*

On exit: must be set to the value of the objective function at **x**, unless you have specified termination of the current problem using INFORM.

4: NSTATE – INTEGER *Input*

On entry: if NSTATE $= 1$ then E05JBF is calling OBJFUN for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

5: IUSER$(*)$ – INTEGER array *User Workspace*
6: RUSER$(*)$ – ***double precision*** array *User Workspace*

OBJFUN is called from E05JBF with the parameters IUSER and RUSER as supplied to E05JBF. You are free to use the arrays IUSER and RUSER to supply information to OBJFUN as an alternative to using COMMON.

7: INFORM – INTEGER *Output*

On exit: should be set to a negative value if you wish to terminate solution of the current problem immediately.

OBJFUN must be declared as EXTERNAL in the (sub)program from which E05JBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: MONIT – SUBROUTINE, supplied by the user. *External Procedure*

MONIT may be used to monitor the minimization process. The default (dummy) monitoring routine E05JBK can be used if no monitoring is required.

Its specification is:

```
      SUBROUTINE OBJFUN (N, NCALL, XBEST, ICOUNT, NINIT, INLIST, LDIN,
     1                   INITPT, NBASKT, XBASKT, NSTATE, IUSER, RUSER,
     2                   INFORM)

      INTEGER            N, NCALL, ICOUNT(6), NINIT, LDIN, INITPT(N),
     1                   NBASKT, NSTATE, IUSER(*), INFORM
      double precision   XBEST(N), INLIST(LDIN,NINIT), XBASKT(N,NBASKT),
     1                   RUSER(*)
```

1: N – INTEGER *Input*

On entry: n, the number of variables, as defined in the call to E05JBF.

2: NCALL – INTEGER *Input*

On entry: the cumulative number of calls to OBJFUN.

3: XBEST(N) – ***double precision*** array *Input*

On entry: the current best point.

4: ICOUNT(6) – INTEGER array *Input*

On entry: an array of integer counters.

ICOUNT(1)

$nboxes$, the current number of sub-boxes.

ICOUNT(2)

$ncloc$, the cumulative number of calls to OBJFUN made in local searches.

ICOUNT(3)

$nloc$, the cumulative number of points used as start points for local searches.

ICOUNT(4)

$nsweep$, the cumulative number of sweeps through levels.

ICOUNT(5)

m, the cumulative number of splits by initialization list.

ICOUNT(6)

s, the current lowest level containing nonsplit boxes.

5: NINIT – INTEGER *Input*

On entry: the maximum over i of the number of points in coordinate i at which to split according to the initialization list INLIST. See also the description of the parameter NUMPTS.

6: INLIST(LDIN,NINIT) – ***double precision*** array *Input*

On entry: the initialization list.

7: LDIN – INTEGER *Input*

On entry: the first dimension of the array INLIST as declared in the (sub)program from which E05JBF is called.

8: INITPT(N) – INTEGER array *Input*

On entry: a pointer to the 'initial point' in INLIST. Element INITPT(i) is the column index in INLIST of the ith coordinate of the initial point.

9: NBASKT – INTEGER *Input*

On entry: the number of points in the 'shopping basket' XBASKT.

10: XBASKT(N,NBASKT) – ***double precision*** array *Input*

On entry: the 'shopping basket' of candidate minima.

11: NSTATE – INTEGER *Input*

On entry: is set by E05JBF to indicate at what stage of the minimization MONIT was called.

NSTATE $= 1$

This is the first time that MONIT has been called.

NSTATE $= -1$

This is the last time MONIT will be called.

NSTATE $= 0$

This is the first *and* last time MONIT will be called.

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

13: RUSER($*$) – ***double precision*** array *User Workspace*

MONIT is called from E05JBF with the parameters IUSER and RUSER as supplied to E05JBF. You are free to use the arrays IUSER and RUSER to supply information to MONIT as an alternative to using COMMON.

14: INFORM – INTEGER *Output*

On exit: should be set to a negative value if you wish to terminate solution of the current problem immediately.

MONIT must be declared as EXTERNAL in the (sub)program from which E05JBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: IBOUND – INTEGER *Input*

On entry: indicates whether the facility for dealing with bounds of special forms is to be used. IBOUND must be set to one of the following values.

IBOUND $= 0$

You will supply **l** and **u** individually.

IBOUND $= 1$

There are no bounds on **x**.

IBOUND $= 2$

There are semi-infinite bounds $0 \leq \mathbf{x}$.

IBOUND $= 3$

There are constant bounds $\mathbf{l} = \ell_1$ and $\mathbf{u} = u_1$.

Constraint: $0 \leq \text{IBOUND} \leq 3$.

5: IINIT – INTEGER *Input*

On entry: selects which initialization method to use.

IINIT $= 0$

Simple initialization (boundary and midpoint), with $\text{NUMPTS}(i) = 3$, $\text{INITPT}(i) = 2$ and $\text{INLIST}(i, 1 : \text{NUMPTS}(i)) = (\text{BL}(i), (\text{BL}(i) + \text{BU}(i))/2, \text{BU}(i))$, for $i = 1, 2, \ldots, \text{N}$.

IINIT $= 1$

Simple initialization (off-boundary and midpoint), with $\text{NUMPTS}(i) = 3$, $\text{INITPT}(i) = 2$ and $\text{INLIST}(i, 1 : \text{NUMPTS}(i)) =$
$((5\text{BL}(i) + \text{BU}(i))/6, (\text{BL}(i) + \text{BU}(i))/2, (\text{BL}(i) + 5\text{BU}(i))/6)$, for $i = 1, 2, \ldots, \text{N}$.

IINIT $= 2$

Initialization using linesearches.

IINIT $= 3$

You are providing your own initialization list.

IINIT $= 4$

Generate a random initialization list.

For more information on methods 2–4 see Section 10.2.

If 'infinite' values (as determined by $infbnd$) are detected by E05JBF when you are using a simple initialization method (IINIT $= 0$ or 1), a safeguarded initialization procedure will be attempted, to avoid overflow.

Suggested value: IINIT $= 0$

Constraint: $0 \leq \text{IINIT} \leq 4$.

6: BL(N) – ***double precision*** array *Input/Output*
7: BU(N) – ***double precision*** array *Input/Output*

On entry: BL is **l**, the array of lower bounds. BU is **u**, the array of upper bounds.

If IBOUND is set to 0, you must set $\text{BL}(i)$ to ℓ_i and $\text{BU}(i)$ to u_i, for $i = 1, 2, \ldots, \text{N}$. If a particular x_i is to be unbounded below, the corresponding $\text{BL}(i)$ should be set to $-infbnd$, where $infbnd$ is the value of the optional parameter **Infinite Bound Size**. Similarly, if a particular x_i is to be unbounded above, the corresponding $\text{BU}(i)$ should be set to $infbnd$.

If IBOUND is set to 1 or 2, arrays BL and BU need not be set on input.

If IBOUND is set to 3, you must set BL(1) to ℓ_1 and BU(1) to u_1. The remaining elements of BL and BU will then be populated by these initial values.

Fixing a variable (that is, setting $\mathrm{BL}(i) = \mathrm{BU}(i)$ for some i) is not permitted by E05JBF.

On exit: the actual arrays of bounds used by E05JBF.

Constraint: $\mathrm{BL}(i) < \mathrm{BU}(i)$ if IBOUND $= 0$ or 3, for $i = 1, 2, \ldots, \mathrm{N}$.

8: NINIT – INTEGER *Input/Output*

On entry: the second dimension of the array INLIST as declared in the (sub)program from which E05JBF is called. NINIT is the maximum over i of the number of points in coordinate i at which to split according to the initialization list INLIST; that is, $\mathrm{NINIT} = \max_i \mathrm{NUMPTS}(i)$.

Internally, E05JBF uses INLIST to determine sets of points along each coordinate direction to which it fits quadratic interpolants. Since fitting a quadratic requires at least three distinct points, this puts a lower bound on NINIT. Furthermore, in the case of initialization by linesearches (IINIT $= 2$) internal storage considerations require that NINIT be at least 192.

On exit: the actual value of NINIT used by E05JBF.

Constraints:

if IINIT $\neq 2$, NINIT ≥ 3;
if IINIT $= 2$, NINIT ≥ 192;
$\mathrm{NINIT} = \max_i \mathrm{NUMPTS}(i)$.

9: INLIST(LDIN,NINIT) – ***double precision*** array *Input/Output*
10: NUMPTS(N) – INTEGER array *Input/Output*
11: INITPT(N) – INTEGER array *Input/Output*

On entry: these three parameters need not be initialized if you wish to use one of the preset initialization methods (IINIT $\neq 3$).

INLIST is the initialization list. Whenever a sub-box in the algorithm is split for the first time (either during the *initialization procedure* or later), the split is done at NUMPTS(i) user-defined values in each coordinate i and at some adaptively chosen intermediate points. You must designate a point stored in INLIST that you wish E05JBF to consider as an 'initial point' for the purposes of the splitting procedure. Call this initial point $\mathbf{x}^*$. If you desire INLIST(i,j) to be x_i^* then you must set INITPT$(i) = j$.

The array sections INLIST$(i, 1 : \mathrm{NUMPTS}(i))$, for $i = 1, 2, \ldots, \mathrm{N}$, must be in ascending order with no repeated entries.

On exit: the actual initialization data used by E05JBF.

Constraints:

$1 \leq \mathrm{INITPT}(i) \leq \mathrm{NINIT}$, for $i = 1, 2, \ldots, \mathrm{N}$;
INITPT$(i, 1 : \mathrm{NUMPTS}i))$ is in ascending order with no repeated entries, for $i = 1, 2, \ldots, \mathrm{N}$;
$\mathrm{BL}(i) \leq \mathrm{INITPT}(i, j) \leq \mathrm{BU}(i)$, for $i = 1, 2, \ldots, \mathrm{N}$ and $j = 1, 2, \ldots, \mathrm{NINIT}$.

12: LDIN – INTEGER *Input*

On entry: the first dimension of the array INLIST as declared in the (sub)program from which E05JBF is called.

Constraint: LDIN $\geq$ N.

13: X(N) – ***double precision*** array *Output*

On exit: if IFAIL $= 0$, contains an estimate of the global optimum (see also Section 7).

14: OBJ – ***double precision*** *Output*

On exit: if IFAIL $= 0$, contains the function value of X.

If you request early termination of E05JBF using INFORM in OBJFUN or the analogous INFORM in MONIT, there is no guarantee that the function value at X equals OBJ.

15: RW(LENRW) – ***double precision*** array *Communication Array*

The array RW **must not** be altered between calls to any of the routines E05JBF, E05JCF, E05JDF, E05JEF, E05JFF, E05JGF, E05JHF, E05JKF and E05JLF.

16: LENRW – INTEGER *Input*

On entry: the dimension of the array RW as declared in the (sub)program from which E05JBF is called.

Constraint: LENRW ≥ 100.

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

Note: the dimension of the array IUSER must be at least 1.

IUSER is not used by E05JBF, but is passed directly to the user-supplied (sub)programs OBJFUN and MONIT. It may be used to pass information to and from those routines.

18: RUSER$(*)$ – ***double precision*** array *User Workspace*

Note: the dimension of the array RUSER must be at least 1.

RUSER is not used by E05JBF, but is passed directly to the user-supplied (sub)programs OBJFUN and MONIT. It may be used to pass information to and from those routines.

19: IFAIL – INTEGER *Input/Output*

On initial entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Chapter P01 for details.

On final exit: IFAIL $= 0$ unless the routine detects an error (see Section 6).

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 parameters 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.**

E05JBF returns with IFAIL $= 0$ if your termination criterion has been met: either a target value has been found to the required relative error (as determined by the values of the optional parameters **Target Objective Value**, **Target Objective Error** and **Target Objective Safeguard**), or the best function value was static for the number of sweeps through levels given by the optional parameter **Static Limit**. The latter criterion is the default.

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 $= 1$

Either the initialization routine E05JAF has not been called, LENRW is less than 100, or the value of N has changed since the last call to E05JAF.

IFAIL $= 2$

An input parameter is invalid. The output message provides more details of the invalid argument.

IFAIL $= 3$

The initialization list contained infinities.

Either the user-supplied initialization list contained infinite values, as determined by the optional parameter **Infinite Bound Size**, or a finite initialization list could not be computed internally. In the latter case you should consider reformulating the bounds on the problem, try providing your own initialization list, use the randomization option (IINIT $= 4$) or vary the value of **Infinite Bound Size**.

IFAIL $= 4$

The division procedure completed but your target value could not be reached.

Despite every sub-box being processed **Splits Limit** times, the target value you provided in **Target Objective Value** could not be found to the tolerances given in **Target Objective Error** and **Target Objective Safeguard**. You could try reducing **Splits Limit** or the objective tolerances.

IFAIL $= 5$

The function evaluations limit was exceeded.

Approximately **Function Evaluations Limit** function calls have been made without your chosen termination criterion being satisfied.

IFAIL $= 6$

You terminated the solver.

You indicated that you wished to halt solution of the current problem by setting INFORM in OBJFUN or INFORM in MONIT to a negative value on exit.

IFAIL $= 7$

An internal error occurred during linesearching. Try relaxing the bounds, rescaling the objective function, or using a different initialization method. If the problem persists, please contact NAG.

7 Accuracy

If IFAIL $= 0$ on exit, then the vector returned in the array X is an estimate of the solution $\mathbf{x}$ whose function value satisfies your termination criterion: the function value was static for **Static Limit** sweeps through levels, or

$$F(\mathbf{x}) - objval \leq \max(objerr \times |objval|, objsfg),$$

where $objval$ is the value of the optional parameter **Target Objective Value**, $objerr$ is the value of the optional parameter **Target Objective Error**, and $objsfg$ is the value of the optional parameter **Target Objective Safeguard**.

8 Further Comments

Local workspace arrays of fixed length are allocated internally by E05JBF. The total size of these arrays amounts to $10\mathrm{N} + smax - 1$ INTEGER elements, where $smax$ is the value of the optional parameter **Splits Limit**, and $\mathrm{NINIT} + 16\mathrm{N} + 3\mathrm{N}^2 + 1$ ***double precision*** elements.

In order to keep track of the regions of the search space that have been visited while looking for a global optimum, E05JBF internally allocates arrays of increasing sizes depending on the difficulty of the problem. Two of the main factors that govern the amount allocated are the number of sub-boxes (call this quantity $nboxes$) and the number of points in the 'shopping basket' (NBASKT). Safe, pessimistic upper bounds on these two quantities are so large as to be impractical. In fact, the worst-case number of sub-boxes for even

the most simple initialization list (when NINIT = 3) grows like n^n as n, the number of variables, increases. Thus E05JBF does not attempt to estimate in advance the final values of $nboxes$ or NBASKT for a given problem. There are a total of 5 INTEGER arrays and $4 + \mathrm{N} + \mathrm{NINIT}$ ***double precision*** arrays whose lengths depend on $nboxes$, and there are a total of 2 INTEGER arrays and $3 + \mathrm{N}$ ***double precision*** arrays whose lengths depend on NBASKT. E05JBF makes a fixed initial guess that the maximum number of sub-boxes required will be 10000 and that the maximum number of points in the 'shopping basket' will be 1000. If ever a greater amount of sub-boxes or more room in the 'shopping basket' is required, E05JBF performs reallocation, doubling the size of the inadequately-sized arrays. Clearly this process requires periods where the original array and its extension exist in memory simultaneously, so that the data within can be copied, which compounds the complexity of E05JBF's memory usage. It is possible (although not likely) that if your problem is particularly difficult to solve, or of a large size (hundreds of variables), you may run out of memory. If this happens an error message will provide more information on the failure.

One array that could be dynamically resized by E05JBF is XBASKT. If the initial attempt to allocate 1000N ***double precision***s for this array fails, MONIT will not be called on exit from E05JBF.

E05JBF performs better if your problem is well-scaled. It is worth trying (by guesswork perhaps) to rescale the problem if necessary, as sensible scaling will reduce the difficulty of the optimization problem, so that E05JBF will take less computer time.

9 Example

This example finds a global minimum of the 'six-hump camelback' function in two dimensions

$$F(x) = \left(4 - 2.1x_1^2 + x_1^4/3\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$$

on the box $[-3, 3] \times [-2, 2]$.

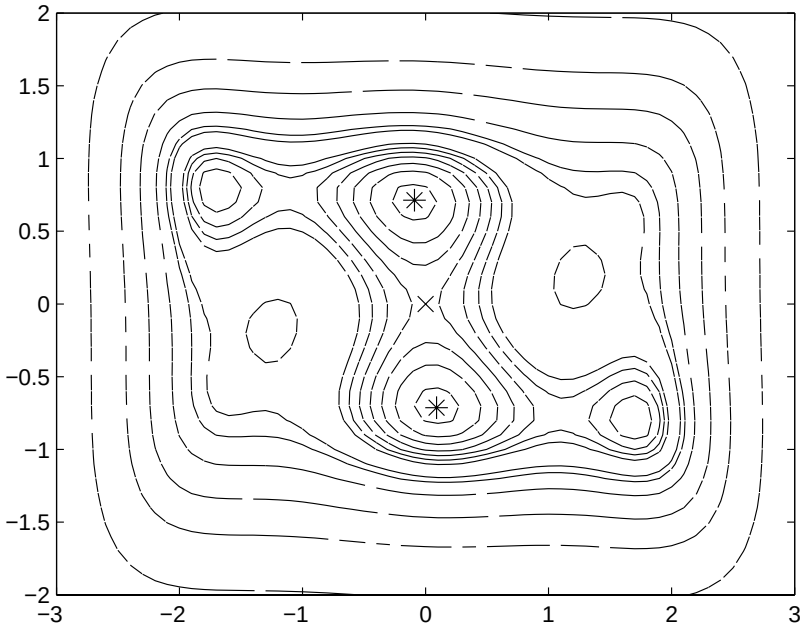

Figure 1
The camelback function F.
The global minima are denoted by $*$, while our start point is labeled with $\times$

The function F has four local minima and two global minima in the given box. The global minima are located at $\pm(0.0898, -0.7126)$, where the function value is -1.0316.

We use default values for all the optional parameters, and we instruct E05JBF to use the simple initialization list corresponding to IINIT = 0. In particular, this will set for us the initial point $(0, 0)$ (marked with a cross in Figure 1).

9.1 Program Text

```
*     E05JBF Example Program Text
*     Mark 22 Release. NAG Copyright 2007.
      IMPLICIT         NONE
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NMAX, LDIN, NINMAX, LENRW
      PARAMETER        (NMAX=2,LDIN=NMAX,NINMAX=192,LENRW=100)
*     .. Local Scalars ..
      DOUBLE PRECISION OBJ
      INTEGER          I, IBOUND, IFAIL, IINIT, N, NINIT
*     .. Local Arrays ..
      DOUBLE PRECISION BL(NMAX), BU(NMAX), INLIST(LDIN,NINMAX),
     +                 RUSER(1), RW(LENRW), X(NMAX)
      INTEGER          INITPT(NMAX), IUSER(1), NUMPTS(NMAX)
*     .. External Subroutines ..
      EXTERNAL         E05JAF, E05JBF, MONIT, OBJFUN
*     .. Executable Statements ..
      CONTINUE
*
      WRITE (NOUT,*) 'E05JBF Example Program Results'
*
*     Skip heading in data file
*
      READ (NIN,*)
*
*     Read N and NINIT from data file
*
      READ (NIN,*) N, NINIT
*
      IF (N.LE.NMAX .AND. NINIT.LE.NINMAX) THEN
*
*        Read IBOUND, BL, and BU from data file
*
         READ (NIN,*) IBOUND
*
         IF (IBOUND.EQ.0) THEN
*
*           Read in the whole of each bound
*
            READ (NIN,*) (BL(I),I=1,N)
            READ (NIN,*) (BU(I),I=1,N)
*
         ELSE IF (IBOUND.EQ.3) THEN
*
*           Bounds are uniform: read in only the first entry of each
*
            READ (NIN,*) BL(1)
            READ (NIN,*) BU(1)
*
         END IF
*
*        Read in IINIT
*
         READ (NIN,*) IINIT
*
*        Call E05JAF to initialize E05JBF
*
         IFAIL = -1
         CALL E05JAF(N,RW,LENRW,IFAIL)
*
*        Solve the problem.
*
         IFAIL = -1
         CALL E05JBF(N,OBJFUN,MONIT,IBOUND,IINIT,BL,BU,NINIT,INLIST,
     +               LDIN,NUMPTS,INITPT,X,OBJ,RW,LENRW,IUSER,RUSER,
     +               IFAIL)
*
         WRITE (NOUT,*)
```

```
         WRITE (NOUT,99999) IFAIL
*
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,99998) OBJ
            WRITE (NOUT,99997) (X(I),I=1,N)
         END IF
*
      END IF
*
      STOP
*
99999 FORMAT (1X,'On exit from E05JBF, IFAIL =',I5)
99998 FORMAT (1X,'Final objective value =',F11.5)
99997 FORMAT (1X,'Global optimum X =',2F9.5)
      END

      SUBROUTINE OBJFUN(N,X,F,NSTATE,IUSER,RUSER,INFORM)
*
*     Routine to evaluate objective function.
*     Mark 22 Release. NAG Copyright 2007.
*
*     .. Parameters ..
      DOUBLE PRECISION  FOUR, THREE
      INTEGER           NOUT
      PARAMETER         (FOUR=4.0D0,THREE=3.0D0,NOUT=6)
*     .. Scalar Arguments ..
      DOUBLE PRECISION  F
      INTEGER           INFORM, N, NSTATE
*     .. Array Arguments ..
      DOUBLE PRECISION  RUSER(*), X(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      DOUBLE PRECISION  X1, X2
*     .. Executable Statements ..
      CONTINUE
*
      INFORM = 0
*
      IF (INFORM.GE.0) THEN
*
*        If INFORM >= 0 then we're prepared to evaluate OBJFUN
*        at the current X
*
         IF (NSTATE.EQ.1) THEN
*
*           This is the first call to OBJFUN
*
            WRITE (NOUT,*)
            WRITE (NOUT,99999)
         END IF
*
         X1 = X(1)
         X2 = X(2)
*
         F = (FOUR-2.1D0*X1**2+X1**4/THREE)*X1**2 + X1*X2 +
     +       (-FOUR+FOUR*X2**2)*X2**2
      END IF
*
      RETURN
*
99999 FORMAT (1X,'(OBJFUN was just called for the first time)')
      END
*
      SUBROUTINE MONIT(N,NCALL,XBEST,ICOUNT,NINIT,INLIST,LDIN,INITPT,
     +                 NBASKT,XBASKT,NSTATE,IUSER,RUSER,INFORM)
*
*     Monitoring routine.
*     Mark 22 Release. NAG Copyright 2007.
*
*     .. Parameters ..
      INTEGER          NOUT
```

```
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      INTEGER          INFORM, LDIN, N, NBASKT, NCALL, NINIT, NSTATE
*     .. Array Arguments ..
      DOUBLE PRECISION INLIST(LDIN,NINIT), RUSER(*), XBASKT(N,NBASKT),
     +                 XBEST(N)
      INTEGER          ICOUNT(6), INITPT(N), IUSER(*)
*     .. Local Scalars ..
      INTEGER          I, J
*     .. Executable Statements ..
      CONTINUE
*
      INFORM = 0
*
      IF (INFORM.GE.0) THEN
*
*        We are going to allow the iterations to continue
*
         IF (NSTATE.EQ.0 .OR. NSTATE.EQ.1) THEN
*
*           When NSTATE.EQ.1, MONIT is called for the first time. When
*           NSTATE.EQ.0, MONIT is called for the first AND last time.
*           Display a welcome message
*
            WRITE (NOUT,*)
            WRITE (NOUT,99999)
            WRITE (NOUT,*)
         END IF
*
         IF (NSTATE.LE.0) THEN
*
*           MONIT is called for the last time.
*
            WRITE (NOUT,99998) ICOUNT(1)
            WRITE (NOUT,99997) NCALL
            WRITE (NOUT,99996) ICOUNT(2)
            WRITE (NOUT,99995) ICOUNT(3)
            WRITE (NOUT,99994) ICOUNT(4)
            WRITE (NOUT,99993) ICOUNT(5)
            WRITE (NOUT,99992) ICOUNT(6)
            WRITE (NOUT,99991) NBASKT
            WRITE (NOUT,99990)
*
            DO 20 I = 1, N
               WRITE (NOUT,99989) I, (XBASKT(I,J),J=1,NBASKT)
   20       CONTINUE
*
            WRITE (NOUT,*)
            WRITE (NOUT,99988)
            WRITE (NOUT,*)
         END IF
*
      END IF
*
      RETURN
*
99999 FORMAT (1X,'*** Begin monitoring information ***')
99998 FORMAT (1X,'Total subboxes =',I5)
99997 FORMAT (1X,'Total function evaluations =',I5)
99996 FORMAT (1X,'Total function evaluations used in local search =',I5)
99995 FORMAT (1X,'Total points used in local search =',I5)
99994 FORMAT (1X,'Total sweeps through levels =',I5)
99993 FORMAT (1X,'Total splits by init. list =',I5)
99992 FORMAT (1X,'Lowest level with nonsplit boxes =',I5)
99991 FORMAT (1X,'Number of candidate minima in the ''shopping basket',
     +       ''' =',I5)
99990 FORMAT (1X,'Shopping basket:')
99989 FORMAT (1X,'XBASKT(',I3,',:) =',(6F9.5))
99988 FORMAT (1X,'*** End monitoring information ***')
      END
```

9.2 Program Data

```
E05JBF Example Program Data
  2   3                                                  : N and NINIT
  0                                                      : IBOUND
  -3.0   -2.0                                            : Lower bounds BL
  3.0   2.0                                              : Upper bounds BU
  0                                                      : IINIT
```

9.3 Program Results

```
 E05JBF Example Program Results

 (OBJFUN was just called for the first time)

 *** Begin monitoring information ***

 Total subboxes =  140
 Total function evaluations =  158
 Total function evaluations used in local search =   92
 Total points used in local search =    7
 Total sweeps through levels =    7
 Total splits by init. list =    5
 Lowest level with nonsplit boxes =    6
 Number of candidate minima in the 'shopping basket' =    2
 Shopping basket:
 XBASKT(  1,:) =  0.08984 -0.08984
 XBASKT(  2,:) = -0.71266  0.71266

 *** End monitoring information ***


 On exit from E05JBF, IFAIL =    0
 Final objective value =   -1.03163
 Global optimum X =  0.08984 -0.71266
```

Note: *the remainder of this document is intended for more advanced users. Section 10 contains a detailed description of the algorithm. This information may be needed in order to understand Section 11, which describes the optional parameters that can be set by calls to E05JCF, E05JDF, E05JEF, E05JFF and/or E05JGF.*

10 Algorithmic Details

Here we summarize the main features of the MCS algorithm used in E05JBF, and we introduce some terminology used in the description of the (sub)program and its arguments. The MCS algorithm is fully described in Huyer and Neumaier (1999).

10.1 Overview

E05JBF searches for a global minimizer using *branching* to recursively split the search space in a nonuniform manner. It divides, or *splits*, the *root box* $[\mathbf{l}, \mathbf{u}]$ into smaller sub-boxes. Each sub-box contains a distinguished *basepoint* at which the objective function is sampled. We shall sometimes say 'the function value of the (sub)box' as shorthand for 'the function value of the basepoint of the (sub)box'. The splitting procedure biases the search in favour of those sub-boxes where low function values are expected.

The global part of the algorithm entails splitting sub-boxes that enclose large unexplored territory, while the local part of the algorithm entails splitting sub-boxes that have good function values. A balance between the global and local part is achieved using a *multilevel* approach, where every sub-box is assigned a *level* $s \in \{0, 1, \ldots, s_{\max}\}$. You can control the value of $s_{\max}$ using the optional parameter **Splits Limit**. A sub-box with level 0 has already been split; a sub-box with level $s_{\max}$ will be split no further. Whenever a sub-box of intermediate level $0 < s < s_{\max}$ is split a descendant will be given the level $s+1$ or $\min\{s+2, s_{\max}\}$. This process is described in more detail in Section 10.2, where the *initialization*

procedure used by E05JBF to produce an initial set of sub-boxes is outlined, and the method by which the algorithm *sweeps* through levels is discussed. Each sweep starts with the sub-boxes at the lowest level, a process thus forming the global part of the algorithm. At each level the sub-box with the best function value is selected for splitting; this forms the local part of the algorithm.

The process by which sub-boxes are split is explained in Section 10.3. It is a variant of the standard coordinate search method: E05JBF splits along a single coordinate at a time, at adaptively chosen points. In most cases one new function evaluation is needed to split a sub-box into two or even three children. Each child is given a basepoint chosen to differ from the basepoint of the parent in at most one coordinate, and safeguards are present to ensure a degree of symmetry in the splits.

If you set the optional parameter **Local Searches** to be 'OFF', then E05JBF puts the basepoints and function values of sub-boxes of maximum level $s_{\max}$ into a 'shopping basket' of candidate minima. Turning **Local Searches** 'ON' (the default setting) will enable local searches to be started from these basepoints before they go into the shopping basket. The local search will go ahead providing the basepoint is not likely to be in the basin of attraction of a previously-found local minimum. The search itself uses a trust region approach, and is explained in Section 10.4: local quadratic models are built by a *triple search*, then a linesearch is made along the direction obtained by minimizing the quadratic on a region where it is a good approximation to the objective function. The quadratic need not be positive definite, so the general nonlinear optimizer E04VHF is used to minimize the model.

10.2 Initialization and Sweeps

Each sub-box is determined by a basepoint $\mathbf{x}$ and an *opposite point* $\mathbf{y}$. We denote such a sub-box by $B[\mathbf{x}, \mathbf{y}]$. The basepoint is allowed to belong to more than one sub-box, is usually a boundary point, and is often a vertex.

An *initialization procedure* produces an initial set of sub-boxes. Whenever a sub-box is split along a coordinate i for the first time (in the initialization procedure or later), the splitting is done at three or more user-defined values $\{x_i^j\}_j$ at which the objective function is sampled, and at some adaptively chosen intermediate points. At least four children are generated. More precisely, we assume that we are given

$$\ell_i \leq x_i^1 < x_i^2 < \cdots < x_i^{L_i} \leq u_i, \quad L_i \geq 3, \quad \text{for } i = 1, 2, \ldots, n$$

and a vector $\mathbf{p}$ that, for each i, locates within $\{x_i^j\}_j$ the ith coordinate of an *initial point* $\mathbf{x}^0$; that is, if $x_i^0 = x_i^j$ for some $j = 1, 2, \ldots, L_i$, then $p_i = j$. A good guess for the global optimum can be used as $\mathbf{x}^0$.

The initialization points and the vectors $\mathbf{L}$ and $\mathbf{p}$ are collectively called the *initialization list* (and sometimes we will refer to just the initialization points as 'the initialization list', whenever this causes no confusion). The initialization data may be input by you, or they can be set to sensible default values by E05JBF: if you provide them yourself, INLIST(i,j) should contain x_i^j, NUMPTS(i) should contain L_i, and INITPT(i) should contain p_i, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, L_i$; if you wish E05JBF to use one of its preset initialization methods, you could choose one of two simple, three-point methods (see Figure 2). If the list generated by one of these methods contains infinite values, attempts are made to generate a safeguarded list using the function subint (which is also used during the splitting procedure, and is described in Section 10.3). If infinite values persist, E05JBF exits with IFAIL = 3. There is also the option to generate an initialization list with the aid of linesearches (by setting IINIT = 2). Starting with the absolutely smallest point in the root box, linesearches are made along each coordinate. For each coordinate, the local minimizers found by the linesearches are put into the initialization list. If there were fewer than three minimizers, they are augmented by close-by values. The final preset initialization option (IINIT = 4) generates a randomized list, so that independent multiple runs may be made if you suspect a global optimum has not been found. Each call to the initialization routine E05JAF resets the initial-state vector for the Wichmann–Hill base-generator that is used. Depending on whether you set the optional parameter **Repeatability** to 'ON' or 'OFF', the random seed is initialized to give a repeatable or nonrepeatable sequence. Then, a random integer between 3 and NINIT is selected, which is then used to determine the number of points to be generated in each coordinate; that is, NUMPTS becomes a constant vector, set to this value. The components of INLIST are then generated, from a uniform distribution on the root box if the box is finite, or else in a safeguarded fashion if any bound is infinite. The array INITPT is set to point to the best point in INLIST.

Given an initialization list (preset or otherwise), E05JBF evaluates F at $\mathbf{x}^0$, and sets the initial estimate of the global minimum, $\mathbf{x}^*$, to $\mathbf{x}^0$. Then, for $i = 1, 2, \ldots, n$, the objective function F is evaluated at $L_i - 1$ points that agree with $\mathbf{x}^*$ in all but the ith coordinate: we obtain L_i pairs $\left(\hat{\mathbf{x}}^j, f_i^j\right)$, for $j = 1, 2, \ldots, L_i$, with: $\mathbf{x}^* = \hat{\mathbf{x}}^{j_1}$, say; with

$$\hat{x}_k^j = \begin{cases} x_k^* & \text{if } k \neq i; \\ x_k^j & \text{otherwise,} \end{cases}$$

for $j \neq j_1$; and with

$$f_i^j = F\left(\hat{\mathbf{x}}^j\right).$$

The point having the smallest function value is renamed $\mathbf{x}^*$ and the procedure is repeated with the next coordinate.

Once E05JBF has a full set of initialization points and function values, it can generate an initial set of sub-boxes. Recall that the *root box* is $B[\mathbf{x}, \mathbf{y}] = [\mathbf{l}, \mathbf{u}]$, having basepoint $\mathbf{x} = \mathbf{x}^0$. The opposite point $\mathbf{y}$ is a corner of $[\mathbf{l}, \mathbf{u}]$ farthest away from $\mathbf{x}$, in some sense. The point $\mathbf{x}$ need not be a vertex of $[\mathbf{l}, \mathbf{u}]$, and $\mathbf{y}$ is entitled to have infinite coordinates. We loop over each coordinate i, splitting the current box along coordinate i into $2L_i - 2$, $2L_i - 1$ or $2L_i$ subintervals with exactly one of the $\hat{x}_i^j$ as endpoints, depending on whether two, one or none of the $\hat{x}_i^j$ are on the boundary. Thus, as well as splitting at $\hat{x}_i^j$, for $j = 1, 2, \ldots, L_i$, we split at additional points z_i^j, for $j = 2, \ldots, L_i$. These additional z_i^j are such that

$$z_i^j = \hat{x}_i^{j-1} + q^m\left(\hat{x}_i^j - \hat{x}_i^{j-1}\right), \quad j = 2, \ldots, L_i,$$

where q is the golden-section ratio $\left(\sqrt{5} - 1\right)/2$, and the exponent m takes the value 1 or 2, chosen so that the sub-box with the smaller function value gets the larger fraction of the interval. Each child sub-box gets as basepoint the point obtained from $\mathbf{x}^*$ by changing x_i^* to the x_i^j that is a boundary point of the corresponding ith coordinate interval; this new basepoint therefore has function value f_i^j. The opposite point is derived from $\mathbf{y}$ by changing y_i to the other end of that interval.

E05JBF can now rank the coordinates based on an estimated variability of F. For each i we compute the union of the ranges of the quadratic interpolant through any three consecutive $\hat{x}_i^j$, taking the difference between the upper and lower bounds obtained as a measure of the variability of F in coordinate i. A vector π is populated in such a way that coordinate i has the π_ith highest estimated variability. For tiebreaks, when the $\mathbf{x}^*$ obtained after splitting coordinate i belongs to two sub-boxes, the one that contains the minimizer of the quadratic models is designated the current sub-box for coordinate $i + 1$.

Boxes are assigned levels in the following manner. The root box is given level 1. When a sub-box of level s is split, the child with the smaller fraction of the golden-section split receives level $s + 2$; all other children receive level $s + 1$. The box with the better function value is given the larger fraction of the splitting interval and the smaller level because then it is more likely to be split again more quickly. We see that after the initialization procedure the first level is empty and the nonsplit boxes have levels $2, \ldots, n + 2$, so it is meaningful to choose $s_{\max}$ much larger than n. Note that the internal structure of E05JBF demands that $s_{\max}$ be at least $n + 3$.

Examples of initializations in two dimensions are given in Figure 2. In both cases the initial point is $\mathbf{x}^0 = (\mathbf{l} + \mathbf{u})/2$; on the left the initialization points are

$$\mathbf{x}^1 = \mathbf{l}, \quad \mathbf{x}^2 = (\mathbf{l} + \mathbf{u})/2, \quad \mathbf{x}^3 = \mathbf{u},$$

while on the right the points are

$$\mathbf{x}^1 = (5\mathbf{l} + \mathbf{u})/6, \quad \mathbf{x}^2 = (\mathbf{l} + \mathbf{u})/2, \quad \mathbf{x}^3 = (\mathbf{l} + 5\mathbf{u})/6.$$

In Figure 2, basepoints and levels after initialization are displayed. Note that these initialization lists correspond to IINIT = 0 and IINIT = 1, respectively.

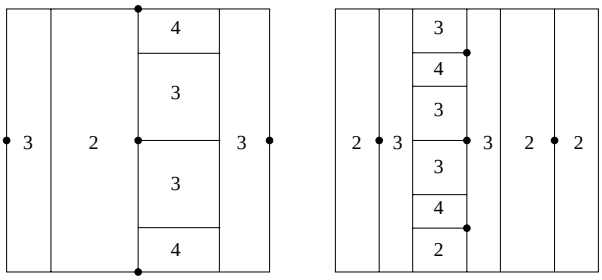

Figure 2
Examples of the initialization procedure

After initialization, a series of *sweeps* through levels is begun. A sweep is defined by three steps:

(i) scan the list of nonsplit sub-boxes. Fill a *record list* **b** according to $b_s = 0$ if there is no box at level s, and with b_s pointing to a sub-box with the lowest function value among all sub-boxes with level s otherwise, for $0 < s < s_{\max}$;

(ii) the sub-box with label b_s is a candidate for splitting. If the sub-box is not to be split, according to the rules described in Section 10.3, increase its level by 1 and update b_{s+1} if necessary. If the sub-box **is** split, mark it so, insert its children into the list of sub-boxes, and update **b** if any child with level s' yields a strict improvement of F over those sub-boxes at level s';

(iii) increment s by 1. If $s = s_{\max}$ then start a new sweep; else if $b_s = 0$ then repeat this step; else go to the previous step.

Clearly, each sweep ends after at most $s_{\max} - 1$ visits of the third step.

10.3 Splitting

Each sub-box is stored by E05JBF as a set of information about the history of the sub-box: the label of its parent, a label identifying which child of the parent it is, etc. Whenever a sub-box $B[\mathbf{x}, \mathbf{y}]$ of level $s < s_{\max}$ is a candidate for splitting, as described in Section 10.2, we recover **x**, **y**, and the number, n_j, of times coordinate j has been split in the history of B. Sub-Box B could be split in one of two ways.

(i) **Splitting by rank**

If $s > 2n(\min n_j + 1)$, the box is always split. The *splitting index* is set to a coordinate i such that $n_i = \min n_j$.

(ii) **Splitting by expected gain**

If $s \leq 2n(\min n_j + 1)$, the sub-box could be split along a coordinate where a maximal gain in function value is expected. This gain is estimated according to a local separable quadratic model obtained by fitting to $2n + 1$ function values. If the expected gain is too small the sub-box is not split at all, and its level is increased by 1.

Eventually, a sub-box that is not eligible for splitting by expected gain will reach level $2n(\min n_j + 1) + 1$ and then be split by rank, as long as $s_{\max}$ is large enough. As $s_{\max} \rightarrow \infty$, the rule for splitting by rank ensures that each coordinate is split arbitrarily often.

Before describing the details of each splitting method, we introduce the procedure for correctly handling splitting at adaptive points and for dealing with unbounded intervals. Suppose we want to split the ith coordinate interval $\{x_i, y_i\}$, where we define $\{x_i, y_i\} = [\min(x_i, y_i), \max(x_i, y_i)]$, for $x_i \in R$ and $y_i \in \bar{R}$, and where **x** is the basepoint of the sub-box being considered. The descendants of the sub-box should shrink sufficiently fast, so we should not split too close to x_i. Moreover, if y_i is large we want the new *splitting value* to **not** be too large, so we force it to belong to some smaller interval $\{\xi', \xi''\}$, determined by

$$\xi'' = \text{subint}(x_i, y_i), \quad \xi' = x_i + (\xi'' - x_i)/10,$$

where the function subint is defined by

$$\text{subint}(x,y) = \begin{cases} \text{sign}(y) & \text{if } 1000|x| < 1 \text{ and } |y| > 1000; \\ 10\text{sign}(y)|x| & \text{if } 1000|x| \geq 1 \text{ and } |y| > 1000|x|; \\ y & \text{otherwise.} \end{cases}$$

10.3.1 Splitting by rank

Consider a sub-box B with level $s > 2n(\min n_j + 1)$. Although the sub-box has reached a high level, there is at least one coordinate along which it has not been split very often. Among the i such that $n_i = \min n_j$ for B, select the splitting index to be the coordinate with the lowest π_i (and hence highest variability rank). 'Splitting by rank' refers to the ranking of the coordinates by n_i and π_i.

If $n_i = 0$, so that B has never been split along coordinate i, the splitting is done according to the initialization list and the adaptively chosen golden-section split points, as described in Section 10.2. Also as covered there, new basepoints and opposite points are generated. The children having the smaller fraction of the golden-section split (that is, those with larger function values) are given level $\min\{s+2, s_{\max}\}$. All other children are given level $s+1$.

Otherwise, B ranges between x_i and y_i in the ith coordinate direction. The splitting value is selected to be $z_i = x_i + 2(\text{subint}(x_i, y_i) - x_i)/3$; we are not attempting to split based on a large reduction in function value, merely in order to reduce the size of a large interval, so z_i may not be optimal. Sub-Box B is split at z_i and the golden-section split point, producing three parts and requiring only one additional function evaluation, at the point $\mathbf{x}'$ obtained from $\mathbf{x}$ by changing the ith coordinate to z_i. The child with the smaller fraction of the golden-section split is given level $\min\{s+2, s_{\max}\}$, while the other two parts are given level $s+1$. Basepoints are assigned as follows: the basepoint of the first child is taken to be $\mathbf{x}$, and the basepoint of the second and third children is the point $\mathbf{x}'$. Opposite points are obtained by changing y_i to the other end of the ith coordinate-interval of the corresponding child.

10.3.2 Splitting by expected gain

When a sub-box B has level $s \leq 2n(\min n_j + 1)$, we compute the optimal splitting index and splitting value from a local separable quadratic used as a simple local approximation of the objective function. To fit this curve, for each coordinate we need two additional points and their function values. Such data may be recoverable from the history of B: whenever the ith coordinate was split in the history of B, we obtained values that can be used for the current quadratic interpolation in coordinate i.

We loop over i; for each coordinate we pursue the history of B back to the root box, and we take the first two points and function values we find, since these are expected to be closest to the current basepoint $\mathbf{x}$. If the current coordinate has not yet been split we use the initialization list. Then we generate a local separable model $e(\xi)$ for $F(\xi)$ by interpolation at $\mathbf{x}$ and the $2n$ additional points just collected:

$$e(\xi) = F(\mathbf{x}) + \sum_{i=1}^{n} e_i(\xi_i).$$

We define the *expected gain* $\hat{e}_i$ in function value when we evaluate at a new point obtained by changing coordinate i in the basepoint, for each i, based on two cases:

(i) $n_i = 0$. We compute the expected gain as

$$\hat{e}_i = \min_{1 \leq j \leq L_i} \{f_i^j\} - f_i^{p_i}.$$

Again, we split according to the initialization list, with the new basepoints and opposite points being as before.

(ii) $n_i > 0$. Now, the ith component of our sub-box ranges from x_i to y_i. Using the quadratic partial correction function

$$e_i(\xi_i) = \alpha_i(\xi_i - x_i) + \beta_i(\xi_i - x_i)^2$$

we can approximate the maximal gain expected when changing x_i only. We will choose the splitting value from $\{\xi', \xi''\}$. We compute

$$\hat{e}_i = \min_{\xi_i \in \{\xi', \xi''\}} e_i(\xi_i)$$

and call z_i the minimizer in $\{\xi', \xi''\}$.

If the expected best function value f_{exp} satisfies

$$f_{\text{exp}} = F(\mathbf{x}) + \min_{1 \le i \le n} \hat{e}_i < f_{\text{best}}, \tag{1}$$

where f_{best} is the current best function value (including those function values obtained by local optimization), we expect the sub-box to contain a better point and so we split it, using as splitting index the component with minimal $\hat{e}_i$. Equation (1) prevents wasting function calls by avoiding splitting sub-boxes whose basepoints have bad function values. These sub-boxes will eventually be split by rank anyway.

We now have a splitting index and a splitting value z_i. The sub-box is split at z_i as long as $z_i \neq y_i$, and at the golden-section split point; two or three children are produced. The larger fraction of the golden-section split receives level $s+1$, while the smaller fraction receives level $\min\{s+2, s_{\max}\}$. If it is the case that $z_i \neq y_i$ and the third child is larger than the smaller of the two children from the golden-section split, the third child receives level $s+1$. Otherwise it is given the level $\min\{s+2, s_{\max}\}$. The basepoint of the first child is set to $\mathbf{x}$, and the basepoint of the second (and third if it exists) is obtained by changing the ith coordinate of $\mathbf{x}$ to z_i. The opposite points are again derived by changing y_i to the other end of the ith coordinate interval of B.

If Equation (1) does not hold, we expect no improvement. We do not split, and we increase the level of B by 1.

10.4 Local Search

The local optimization algorithm used by E05JBF uses linesearches along directions that are determined by minimizing quadratic models, all subject to bound constraints. Triples of vectors are computed using *coordinate searches* based on linesearches. These triples are used in *triple search* procedures to build local quadratic models for F. A trust-region-type approach to minimize these models is then carried out, and more information about the coordinate search and the triple search can be found in Huyer and Neumaier (1999).

The local search starts by looking for better points without being too local, by making a triple search using points found by a coordinate search. This yields a new point and function value, an approximation of the gradient of the objective, and an approximation of the Hessian of the objective. Then, the quadratic model for F is minimized over a box $[-\mathbf{d}, \mathbf{d}]$, with the solution to that minimization problem then being used as a linesearch direction to minimize the objective. A measure r is computed to quantify the predictive quality of the quadratic model.

The third stage is the checking of termination criteria: the local search stops if more than $loclim$ visits to this part of the local search have occurred, where $loclim$ is the value of the optional parameter **Local Searches Limit**; it stops if the limit on function calls has been exceeded (see the description of the optional parameter **Function Evaluations Limit**); and the final criterion checks if no improvement can be made to the function value, or whether the approximated gradient $\mathbf{g}$ is small, in the sense that

$$|\mathbf{g}|^T \max(|\mathbf{x}|, |\mathbf{x}_{\text{old}}|) < loctol(f - f_0).$$

The vector $\mathbf{x}_{\text{old}}$ is the best point at the start of the current loop in this iterative local-search procedure, the constant $loctol$ is the value of the optional parameter **Local Searches Tolerance**, f is the objective value at $\mathbf{x}$, and f_0 is the smallest function value found by the initialization procedure.

Next, E05JBF attempts to move away from the boundary, if any components of the current point lie there, using linesearches along the offending coordinates. Local searches are terminated if no improvement could be made.

The fifth stage carries out another triple search, but this time it does not use points from a coordinate search, rather points lying within the trust-region box $[-\mathbf{d}, \mathbf{d}]$ are taken.

The final stage modifies the trust-region box to be bigger or smaller, depending on the quality of the quadratic model, minimizes the new quadratic model on that box, and does a linesearch in the direction of the minimizer. The value of r is updated using the new data, and then we go back to the third stage (checking of termination criteria).

The Hessians of the quadratic models generated by the local search may not be positive definite, so E05JBF uses the general nonlinear optimizer E04VHF to minimize the models.

11 Optional Parameters

Several optional parameters in E05JBF define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of E05JBF 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.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters. A complete list of optional parameters and their default values is given in Section 11.1.

Optional parameters may be specified by calling one, or more, of the routines E05JCF, E05JDF, E05JEF, E05JFF and E05JGF before a call to E05JBF.

E05JCF reads options from an external options file, with `Begin` and `End` as the first and last lines respectively, and with each intermediate line defining a single optional parameter. For example,

```
Begin
  Static Limit = 50
End
```

The call

```
CALL E05JCF (IOPTS, RW, LENRW, IFAIL)
```

can then be used to read the file on unit IOPTS. IFAIL will be zero on successful exit. E05JCF should be consulted for a full description of this method of supplying optional parameters.

E05JDF, E05JEF, E05JFF or E05JGF can be called to supply options directly, one call being necessary for each optional parameter. E05JDF, E05JEF, E05JFF or E05JGF should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by you are set to their default values. Valid values of optional parameters specified by you are unaltered by E05JBF and so remain in effect for subsequent calls to E05JBF, unless altered by you.

11.1 Optional Parameter Checklist and Default Values

The following list gives the valid options. For each option, we give the keywords, the symbolic name, the default value, and the allowed range. A definition for each option can be found in Section 11.2. Option names are case-insensitive and must be provided in full; abbreviations are not recognized. The letter a denotes an option that takes an 'ON' or 'OFF' value; the letters i and r denote INTEGER and ***double precision*** values required with certain options, respectively. The symbol ϵ is a generic notation for ***machine precision*** (see X02AJF), and the variable $infbnd$ holds the value of **Infinite Bound Size**.

Optional Parameters	**Symbolic Name**	**Default Values**
Allowed Range		
Defaults		
Function Evaluations Limit	*nf*	Default $= 50n^2$
$nf > 0$		
Infinite Bound Size	*infbnd*	Default $= 10^{20}$
$infbnd \geq 10^{20}$		
List		See **Nolist**

Local Searches	*lcsrch*	Default = 'ON'
$lcsrch =$ 'ON' or 'OFF'		
Local Searches Limit	*loclim*	Default = 50
$loclim > 0$		
Local Searches Tolerance	*loctol*	Default = 2ϵ
$loctol \geq 2\epsilon$		
Maximize		See **Minimize**
Minimize		Default
Nolist		Default
Repeatability	*repeat*	Default = 'OFF'
$repeat =$ 'ON' or 'OFF'		
Splits Limit	*smax*	Default = $5n + 10$
$smax > n + 2$		
Static Limit	*stclim*	Default = $3n$
$stclim > 0$		
Target Objective Error	*objerr*	Default = 1.0D2
$objerr \geq 2\epsilon$		
Target Objective Safeguard	*objsfg*	Default = 1.0D6
$objsfg \geq 2\epsilon$		
Target Objective Value	*objval*	

11.2 Description of the Optional Parameters

Defaults

This special keyword is used to reset all optional parameters to their default values.

Function Evaluations Limit i Default = $50n^2$

This puts an approximate limit on the number of function calls allowed. The total number of calls made is checked at the top of an internal iteration loop, so it is possible that a few calls more than nf may be made.

Constraint: $nf > 0$.

Infinite Bound Size r Default = 10^{20}

This defines the 'infinite' bound $infbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $infbnd$ will be regarded as ∞ (and similarly any lower bound less than or equal to $-infbnd$ will be regarded as $-\infty$).

Constraint: $infbnd \geq 10^{20}$.

Local Searches a Default = 'ON'

If you want to try to accelerate convergence of E05JBF by starting local searches from candidate minima, you will require $lcsrch$ to be 'ON'.

Constraint: $lcsrch =$ 'ON' or 'OFF'.

Local Searches Limit i Default = 50

This defines the maximal number of iterations to be used in the trust-region loop of the local-search procedure.

Constraint: $loclim > 0$.

Local Searches Tolerance r Default $= 2\epsilon$

The value of *loctol* is the multiplier used during local searches as a stopping criterion for when the approximated gradient is small, in the sense described in Section 10.4.

Constraint: $loctol \geq 2\epsilon$.

Minimize Default
Maximize

These keywords specify the required direction of optimization.

Nolist Default
List

These options control the echoing of each optional parameter specification as it is supplied. **List** turns printing on, **Nolist** turns printing off. The output is sent to the current advisory message unit (as defined by X04ABF).

Repeatability a Default = 'OFF'

For use with random initialization lists (IINIT = 4). When set to 'ON', an internally-initialized random seed is stored in the array RW for use in subsequent calls to E05JBF.

Constraint: $repeat =$ 'ON' or 'OFF'.

Splits Limit i Default $= 5n + 10$

Along with INLIST, this defines a limit on the number of times the root box will be split along any single coordinate direction. If **Local Searches** is 'OFF' you may find the default value to be too small.

Constraint: $smax > n + 2$.

Static Limit i Default $= 3n$

As the default termination criterion, computation stops when the best function value is static for $stclim$ sweeps through levels. This parameter is ignored if you have specified a target value to reach in **Target Objective Value**.

Constraint: $stclim > 0$.

Target Objective Error r Default = 1.0D2

If you have given a target objective value to reach in $objval$ (the value of the optional parameter **Target Objective Value**), $objerr$ sets your desired relative error (from above if **Minimize** is set, from below if **Maximize** is set) between OBJ and $objval$, as described in Section 7. See also the description of the optional parameter **Target Objective Safeguard**.

Constraint: $objerr \geq 2\epsilon$.

Target Objective Safeguard r Default = 1.0D6

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 Value r

This parameter may be set if you wish E05JBF to use a specific value as the target function value to reach during the optimization. Setting $objval$ overrides the default termination criterion determined by the optional parameter **Static Limit**.
