NAG Toolbox for Matlab

e05jb

1 Purpose

e05jb is designed to find the global minimum or maximum of an arbitrary function, subject to simple bound-constraints using a multi-level 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.

e05jb uses **forward communication** for evaluating the objective function.

The initialization function e05ja **must** have been called before calling e05jb.

2 Syntax

```
[bl, bu, list, numpts, initpt, x, obj, comm, user, ifail] =
e05jb(objfun, ibound, iinit, bl, bu, list, numpts, initpt, monit, comm,
'n', n, 'sdlist', sdlist, 'user', user)
```

Before calling e05jb, or any of the option-setting or option-getting functions e05jd, e05je, e05jf, e05jg, e05jh, e05jj, e05jk or e05jl, e05ja **must** be called.

The contents of the array **comm** and the value of the parameter **n must not** be altered between calls of the functions e05ja, e05jb, e05jd, e05je, e05jf, e05jg, e05jh, e05jj, e05jk or e05jl.

3 Description

e05jb 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:

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

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 function calling e05jb is:

```
[comm, ifail] = e05ja(n);
[comm, ifail] = e05jd(optstr, comm);
[..., ifail] = e05jb(objfun, ...);
```

where e05jd sets the optional parameter and value specified in **optstr**.

The initialization function e05ja does not need to be called before each invocation of e05jb, so long as the problem size, **n**, remains unchanged. You should be aware that a call to the initialization function 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 **comm** will be destroyed.

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

The method used by e05jb is based on MCS, the Multi-level 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 Multi-level Coordinate Search *Journal of Global Optimization* **14** 331–355

5 Parameters

5.1 Compulsory Input Parameters

1: **objfun – string containing name of m-file**

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

```
[f, user, inform] = objfun(n, x, nstate, user)
```

Input Parameters

1: **n – int32 scalar**

n, the number of variables.

2: **x(n) – double array**

x, the vector at which the objective function is to be evaluated.

3: **nstate – int32 scalar**

If **nstate** $= 1$ then e05jb 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.

4: **user – Any MATLAB object**

objfun is called from e05jb with **user** as supplied to e05jb

Output Parameters

1: **f – double scalar**

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

2: **user – Any MATLAB object**

objfun is called from e05jb with **user** as supplied to e05jb

3: **inform – int32 scalar**

Must be set to a value describing the action to be taken by the solver on return from **objfun**: if the value is negative the solution of the current problem will terminate immediately; otherwise, computations will continue.

2: **ibound – int32 scalar**

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 $0 \leq$ **x**.

ibound $= 3$
There are constant bounds **l** $= \ell_1$ and **u** $= u_1$.

Constraint: $0 \leq$ **ibound** ≤ 3.

3: **iinit – int32 scalar**

Selects which initialization method to use.

iinit $= 0$
Simple initialization (boundary and midpoint), with **numpts**$(i) = 3$, **initpt**$(i) = 2$ and **list**$(i, 1 :$ **numpts**$(i)) =$ (**bl**(i), (**bl**$(i) +$ **bu**$(i))/2$, **bu**$(i))$, for $i = 1, 2, \ldots,$ **n**.

iinit $= 1$
Simple initialization (off-boundary and midpoint), with **numpts**$(i) = 3$, **initpt**$(i) = 2$ and **list**$(i, 1 :$ **numpts**$(i)) =$ $((5$**bl**$(i) +$ **bu**$(i))/6$, (**bl**$(i) +$ **bu**$(i))/2$, (**bl**$(i) + 5$**bu**$(i))/6)$, for $i = 1, 2, \ldots,$ **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 10.1.

If 'infinite' values (as determined by the value of the optional parameter Infinite Bound Size) are detected by e05jb 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** $= 0$

Default: **iinit** $= 0$

Constraint: $0 \leq$ **iinit** ≤ 4.

4: **bl**(**n**) **– double array**

5: **bu**(**n**) **– double array**

n, the dimension of the array, must satisfy the constraint **n** > 0.

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 **bl**(i) to ℓ_i and **bu**(i) to u_i, for $i = 1, 2, \ldots,$ **n**. If a particular x_i is to be unbounded below, the corresponding **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 **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 **bl**$(i) =$ **bu**(i) for some i) is not permitted by e05jb.

Constraints:

if **ibound** $= 0$ or 3, **bl**$(i) <$ **bu**(i);

for $i = 1, 2, \ldots,$ **n**.

6: **list**(**n**,**sdlist**) – **double array**
7: **numpts**(**n**) – **int32 array**
8: **initpt**(**n**) – **int32 array**

n, the first dimension of the array, must satisfy the constraint $\mathbf{n} > 0$.

These three parameters need not be set on entry if you wish to use one of the preset initialization methods (**iinit** $\neq 3$).

list is the 'initialization list': whenever a sub-box in the algorithm is split for the first time (either during the *initialization procedure* or later), for each coordinate i the split is done at the first **numpts**(i) values in row i of **list**, as well as at some adaptively chosen intermediate points.

You must designate a point stored in **list** that you wish e05jb to consider as an 'initial point' for the purposes of the splitting procedure. Call this initial point $\mathbf{x}^*$. If you desire **list**(i,j) to be x_i^* then you must set **initpt**$(i) = j$.

The array sections **list**$(i, 1 : \mathbf{numpts}(i))$, for $i = 1, 2, \ldots, \mathbf{n}$, must be in ascending order with each entry being distinct. In this context, 'distinct' should be taken to mean relative to the safe-range parameter (see x02am).

Constraints:

$1 \leq \mathbf{initpt}(i) \leq \mathbf{sdlist}$, for $i = 1, 2, \ldots, \mathbf{n}$;
list$(i, 1 : \mathbf{numpts}(i))$ is in ascending order with each entry being distinct, for $i = 1, 2, \ldots, \mathbf{n}$;
$\mathbf{bl}(i) \leq \mathbf{list}(i,j) \leq \mathbf{bu}(i)$, for $i = 1, 2, \ldots, \mathbf{n}$ and $j = 1, 2, \ldots, \mathbf{numpts}(i)$.

9: **monit – string containing name of m-file**

monit may be used to monitor the minimization process. It is invoked upon every successful completion of the procedure in which a sub-box is considered for splitting. It will also be called just before e05jb exits if that splitting procedure was not successful.

If no monitoring is required, **monit** may be string `'e05jbk'`

```
[user, inform] = monit(n, ncall, xbest, icount, ninit, list, numpts,
initpt, nbaskt, xbaskt, boxl, boxu, nstate, user)
```

Input Parameters

1: **n – int32 scalar**

n, the number of variables.

2: **ncall – int32 scalar**

The cumulative number of calls to **objfun**.

3: **xbest**(**n**) – **double array**

The current best point.

4: **icount**(6) – **int32 array**

An array of 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 non-split boxes.

5: **ninit – int32 scalar**

The maximum over i of the number of points in coordinate i at which to split according to the initialization list **list**. See also the description of the parameter **numpts**.

6: **list**$($**n**,**ninit**$)$ **– double array**

The initialization list.

7: **numpts**$($**n**$)$ **– int32 array**

The number of points in each coordinate at which to split according to the initialization list **list**.

8: **initpt**$($**n**$)$ **– int32 array**

A pointer to the 'initial point' in **list**. Element **initpt**(i) is the column index in **list** of the ith coordinate of the initial point.

9: **nbaskt – int32 scalar**

The number of points in the 'shopping basket' **xbaskt**.

10: **xbaskt**$($**n**,**nbaskt**$)$ **– double array**

The 'shopping basket' of candidate minima.

11: **boxl**$($**n**$)$ **– double array**

The array of lower bounds of the current search box.

12: **boxu**$($**n**$)$ **– double array**

The array of upper bounds of the current search box.

13: **nstate – int32 scalar**

Is set by e05jb 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.

14: **user – Any MATLAB object**

monit is called from e05jb with **user** as supplied to e05jb

Output Parameters

1: **user – Any MATLAB object**

monit is called from e05jb with **user** as supplied to e05jb

2: **inform – int32 scalar**

Must be set to a value describing the action to be taken by the solver on return from **monit**: if the value is negative the solution of the current problem will terminate immediately; otherwise, computations will continue.

10: **comm**(*lcomm*) **– double array**

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: The dimension of the arrays **bl**, **bu**, **numpts**, **initpt** and the first dimension of the array **list**. (An error is raised if these dimensions are not equal.)

n, the number of variables.

Constraint: $\mathbf{n} > 0$.

2: **sdlist – int32 scalar**

Default: The second dimension of the array **list**.

sdlist is, at least, the maximum over i of the number of points in coordinate i at which to split according to the initialization list **list**; that is, $\mathbf{sdlist} \geq \max_i \mathbf{numpts}(i)$.

Internally, e05jb uses **list** 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 **sdlist**. Furthermore, in the case of initialization by linesearches (**iinit** $= 2$) internal storage considerations require that **sdlist** be at least 192, but not all of this space may be used.

Constraints:

if $\mathbf{iinit} \neq 2$, $\mathbf{sdlist} \geq 3$;
if $\mathbf{iinit} = 2$, $\mathbf{sdlist} \geq 192$;
if $\mathbf{iinit} = 3$, $\mathbf{sdlist} \geq \max_i \mathbf{numpts}(i)$.

3: **user – Any MATLAB object**

user is not used by e05jb, but is passed to **objfun** and **monit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

lcomm

5.4 Output Parameters

1: **bl**(**n**) **– double array**
2: **bu**(**n**) **– double array**

Unless **ifail** $= 1$ or 2 on exit, **bl** and **bu** are the actual arrays of bounds used by e05jb.

3: **list**(**n**,**sdlist**) **– double array**

4: **numpts**(**n**) **– int32 array**

5: **initpt**(**n**) **– int32 array**

Unless **ifail** = 1, 2 or −999 on exit, the actual initialization data used by e05jb. If you wish to monitor the contents of **list** you are advised to do so solely through **monit**, not through the output value here.

6: **x**(**n**) **– double array**

If **ifail** = 0, contains an estimate of the global optimum (see also Section 7).

7: **obj – double scalar**

If **ifail** = 0, contains the function value of **x**.

If you request early termination of e05jb using **inform** in **objfun** or the analogous **inform** in **monit**, there is no guarantee that the function value at **x** equals **obj**.

8: **comm**(*lcomm*) **– double array**

comm must not be altered between calls to any of the functions e05jb, e05jd, e05je, e05jf, e05jg, e05jh, e05jj, e05jk and e05jl.

9: **user – Any MATLAB object**

10: **ifail – int32 scalar**

ifail = 0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

Either the initialization function e05ja has not been called, *lcomm* is less than 100, or the value of **n** has changed since the last call to e05ja.

ifail = 2

An input parameter is invalid. If **ifail** = 0 or −1 on entry, 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 *smax* times (where *smax* is the value of the optional parameter Splits Limit), the target value you provided via the optional parameter Target Objective Value could not be found to the tolerances given in the optional parameters Target Objective Error and Target Objective Safeguard. You could try increasing Splits Limit or the objective tolerances.

ifail $= 5$

The function evaluations limit was exceeded.

Approximately *nf* function calls (where *nf* is the value of the optional parameter Function Evaluations Limit) 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. If **ifail** $= 0$ or -1 on entry to e05jb.

ifail $= 7$

No further progress could be made on your problem. Try rescaling the objective function, relaxing the bounds, or using a different initialization method.

ifail $= -999$

Internal memory allocation failed.

7 Accuracy

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

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

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

For each invocation of e05jb, local workspace arrays of fixed length are allocated internally. The total size of these arrays amounts to $10\mathbf{n} + \mathit{smax} - 1$ integer elements, where *smax* is the value of the optional parameter Splits Limit, and $2\mathbf{sdlist} + 18\mathbf{n} + 3\mathbf{n}^2 + 1$ double elements. In addition, if you are using randomized initialization lists (see the description of the parameter **iinit**), a further 21 integer elements are allocated internally.

In order to keep track of the regions of the search space that have been visited while looking for a global optimum, e05jb 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' (the parameter **nbaskt** on entry to **monit**). 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** $= 3$ on entry to **monit**) grows like n^n as n, the number of variables, increases. Thus e05jb 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 $4 + \mathbf{n} + \mathbf{ninit}$ double arrays whose lengths depend on *nboxes*, and there are a total of 2 integer arrays and $3 + \mathbf{n}$ double arrays whose lengths depend on **nbaskt**. e05jb 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, e05jb performs reallocation, usually 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 e05jb'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.

One array that could be dynamically resized by e05jb is the 'shopping basket' (**xbaskt** on entry to **monit**). If the initial attempt to allocate 1000**n** doubles for this array fails, **monit** will not be called on exit from e05jb.

e05jb 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 e05jb will take less computer time.

9 Example

e05jb_monitor.m

```
function                          [user,inform]                          =
e05jb_monitor(n,ncall,xbest,icount,ninit,list,numpts,initpt,nbaskt,
xbaskt,boxl,boxu,nstate,user)

inform = int32(0);

if (nstate == 0 || nstate == 1)
    disp(sprintf('\n'))
    disp('*** Begin monitoring information ***');
    disp(sprintf('\n'))
end

if (nstate <= 0)
    disp(['Total sub-boxes = ' num2str(icount(1))]);
    disp(['Total function evaluations = ' num2str(ncall)]);
      disp(['Total function evaluations used in local searches = '
num2str(icount(2))]);
    disp(['Total points used in local search = ' num2str(icount(3))]);
    disp(['Total sweeps through levels = ' num2str(icount(4))]);
    disp(['Total splits by init. list = ' num2str(icount(5))]);
    disp(['Lowest level with nonsplit boxes = ' num2str(icount(6))]);
     disp(['Number of candidate minima in the ''shopping basket'' = '
num2str(nbaskt)]);
    disp('Shopping basket:');
    disp(xbaskt);
    disp(sprintf('\n'))
    disp('*** End monitoring information ***');
    disp(sprintf('\n'))
end
```

e05jb_objective.m

```
function [f,user,inform] = e05jb_objective(n,x,nstate,user)

if (n==2)
    inform = int32(0);
else
    inform = int32(-1);
end

if (inform >= 0)

    % We're prepared to evaluate the objective at this point

    if (nstate == 1)
        disp(sprintf('\n'));
        disp('(OBJFUN was just called for the first time)');
    end

    f = peaks(x(1), x(2));
end
```

```
% Problem data for peaks function
prob = 'peaks';
xres = 100;
yres = 100;

bl = [-3; -3];
bu = -bl;
fglob = -6.55; % Approx.
xglob = [0.23; -1.63]; % Approx.



% Initialize e05jb
n = int32(length(bl));
[comm, ifail] = e05ja(n);

if (ifail == 0)

    % Vanilla call.
    disp(sprintf('\n'));
    disp('Solve with no options or init.-list data');

    ibound = int32(0);             % All bounds will be given;
    iinit = int32(0);              % Default initialization method;
     list = zeros(n,3);              % Only need to _declare_ the init.-
list
       numpts = zeros(n, 1, 'int32'); % data: these will be _set_
internally.
    initpt = zeros(n, 1, 'int32');

    [bl, bu, listOut, numptsOut, initptOut, ...
     xbest, obj, comm, userOut, ifail] = ...
        e05jb('e05jb_objective', ibound, iinit, bl, bu, list, ...
              numpts, initpt, 'e05jb_monitor', comm);

    disp(['e05jb (no options) exited with ifail = ' num2str(ifail)]);

    if (ifail == 0)
        disp('xbest:');
        disp(xbest);
        disp(['obj = ' num2str(obj)]);
    end

    % Set some options. No need to reinitialize: n hasn't changed, and we
    % didn't set any options above.
    disp(sprintf('\n'));
    disp('Solve with options and init.-list data');

    % Echo the setting of opt. params.
    comm = e05jd('List', comm);

    comm = e05jd('Function Evaluations Limit = 100000', comm);
    comm = e05jf('Static Limit', 3*n, comm);

    % Increase infbnd by factor of 10.
    infbnd = e05jl('Infinite Bound Size', comm);
    comm = e05jg('Infinite Bound Size', 10*infbnd, comm);

    comm = e05je('Local Searches', 'on', comm);

    % Set the initialization-list data.
     iinit = int32(3);                          % We're providing the data
this time:
    list = zeros(n, 3);
    list(:, 1) = bl; list(:, 3) = bu;
    list(:, 2) = [-1; 0];
     numpts = int32(3)*ones(n, 1, 'int32'); % 3 splitting points for each
dim;
    initpt = int32(2)*ones(n, 1, 'int32'); % 2nd pt in each row to be the
'init.' pt.
```

```
    [bl, bu, listOut, numptsOut, initptOut, ...
     xbest, obj, comm, userOut, ifail] = ...
        e05jb('e05jb_objective', ibound, iinit, bl, bu, list, ...
              numpts, initpt, 'e05jb_monitor', comm);

    disp(['e05jb (options) exited with ifail = ' num2str(ifail)]);

    if (ifail == 0)
        disp('xbest:');
        disp(xbest);
        disp(['obj = ' num2str(obj)]);
    end

end
```

```
Solve with no options or init.-list data


(OBJFUN was just called for the first time)


*** Begin monitoring information ***


Total sub-boxes = 228
Total function evaluations = 196
Total function evaluations used in local searches = 87
Total points used in local search = 13
Total sweeps through levels = 12
Total splits by init. list = 5
Lowest level with nonsplit boxes = 7
Number of candidate minima in the 'shopping basket' = 2
Shopping basket:
   -1.3474    0.2283
    0.2045   -1.6255


*** End monitoring information ***


e05jb (no options) exited with ifail = 0
xbest:
    0.2283
   -1.6255
obj = -6.5511


Solve with options and init.-list data
      Function Evaluations Limit = 100000
      Static Limit               6
      Infinite Bound Size    1.1579208923731620E+78
      Local Searches on


(OBJFUN was just called for the first time)


*** Begin monitoring information ***


Total sub-boxes = 146
Total function evaluations = 169
Total function evaluations used in local searches = 102
Total points used in local search = 7
Total sweeps through levels = 7
Total splits by init. list = 5
Lowest level with nonsplit boxes = 4
Number of candidate minima in the 'shopping basket' = 2
Shopping basket:
```

```
    0.2283   -1.3474
   -1.6255    0.2045


*** End monitoring information ***


e05jb (options) exited with ifail = 0
xbest:
    0.2283
   -1.6255
obj = -6.5511
```

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 e05jd, e05je, e05jf and/or e05jg.*

10 Algorithmic Details

Here we summarize the main features of the MCS algorithm used in e05jb, and we introduce some terminology used in the description of the function and its arguments. The MCS algorithm is fully described in Huyer and Neumaier (1999).

10.1 Initialization and Sweeps

Each sub-box is determined by a basepoint **x** and an *opposite point* **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 $\left\{x_i^j\right\}_j$ 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

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

and a vector **p** that, for each i, locates within $\left\{x_i^j\right\}_j$ 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 **L** and **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 e05jb: if you provide them yourself, **list**(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 e05jb to use one of its preset initialization methods, you could choose one of two simple, three-point methods (see Figure 1). If the list generated by one of these methods contains infinite values, attempts are made to generate a safeguarded list using the function subint(x, y) (which is also used during the splitting procedure, and is described in Section 10.2). If infinite values persist, e05jb 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 function e05ja 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 non-repeatable sequence. Then, a random integer between 3 and **sdlist** 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 **list** 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 **list**.

Given an initialization list (preset or otherwise), e05jb 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 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, for $j \neq j_1$,

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

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 e05jb 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

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

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.

e05jb 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 non-split 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 e05jb demands that $s_{\max}$ be at least $n + 3$.

Examples of initializations in two dimensions are given in Figure 1. In both cases the initial point is $\mathbf{x}^0 = (\mathbf{l} + \mathbf{u})/2$; on the left the initialization points are

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

while on the right the points are

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

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

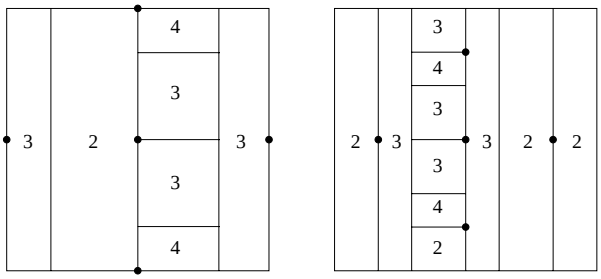

Figure 1
Examples of the initialization procedure

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

(i) scan the list of non-split 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 10.2, 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 displaying monitoring information and start a new sweep; else if $b_s = 0$ then repeat this step; else display monitoring information and go to the previous step.

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

10.2 Splitting

Each sub-box is stored by e05jb 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 10.1, 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 $s > 2n\big(\min n_j + 1\big)$, 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 $s \leq 2n\big(\min n_j + 1\big)$, 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 $2n\big(\min n_j + 1\big) + 1$ and then be split by rank, as long as $s_{\max}$ is large enough. As $s_{\max} \to \infty$, 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 $\square\{x_i, y_i\}$, where we define $\square\{x_i, y_i\} = [\min(x_i, y_i), \max(x_i, y_i)]$, 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 $\square\{\xi', \xi''\}$, determined by

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

where the function subint is defined by

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

10.2.1 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 10.1. 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 **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 **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.

10.2.2 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 **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 **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

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

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 $\square\{\xi', \xi''\}$. We compute

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

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

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.

10.3 Local Search

The local optimization algorithm used by e05jb 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 small box, 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 will stop 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. If that is not the case, it will stop if the limit on function calls has been exceeded (see the description of the optional parameter Function Evaluations Limit). 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

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

The vector $\mathbf{x}_{\mathrm{old}}$ 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, e05jb 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 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 e05jb uses the general nonlinear optimizer e04vh to minimize the models.

11 Optional Parameters

Several optional parameters in e05jb define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of e05jb 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 functions e05jd, e05je, e05jf and e05jg before a call to e05jb.

All optional parameters not specified by you are set to their default values. Valid values of optional parameters specified by you are unaltered by e05jb and so remain in effect for subsequent calls to e05jb, 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 real values required with certain options, respectively. The symbol ϵ is a generic notation for ***machine precision*** (see x02aj), the symbol $r_{\max}$ stands for the largest positive model number (see x02al), and the symbol d stands for the maximum number of decimal digits that can be represented (see x02be).

Optional Parameter	**Symbolic Name**	**Default Value**	**Allowed Range**
Defaults			
Function Evaluations Limit	*nf*	Default $= 100n^2$	$nf > 0$
Infinite Bound Size	*infbnd*	Default $= r_{\max}^{\frac{1}{4}}$	$r_{\max}^{\frac{1}{4}} \leq infbnd \leq r_{\max}^{\frac{1}{2}}$
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 $= 2\epsilon$	$loctol \geq 2\epsilon$
Maximize		See Minimize	
Minimize		Default	

Nolist		Default	
Repeatability	*repeat*	Default = ' OFF '	*repeat* = 'ON' or 'OFF'
Splits Limit	*smax*	Default = floor$(d(n+2)/3)$	*smax* $> n+2$
Static Limit	*stclim*	Default $= 3n$	*stclim* > 0
Target Objective Error	*objerr*	Default $= \epsilon^{\frac{1}{4}}$	*objerr* $\geq 2\epsilon$
Target Objective Safeguard	*objsfg*	Default $= \epsilon^{\frac{1}{2}}$	*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 *(nf)* i Default $= 100n^2$

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 *(infbnd)* r Default $= r_{\max}^{\frac{1}{4}}$

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: $r_{\max}^{\frac{1}{4}} \leq$ *infbnd* $\leq r_{\max}^{\frac{1}{2}}$.

Local Searches *(lcsrch)* a Default = ' ON '

If you want to try to accelerate convergence of e05jb by starting local searches from candidate minima, you will require *lcsrch* to be 'ON'.

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

Local Searches Limit *(loclim)* 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 *(loctol)* 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 10.3.

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 x04ab).

Repeatability *(repeat)* 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 **comm** for use in subsequent calls to e05jb.

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

Splits Limit *(smax)* i Default $= \text{floor}(d(n+2)/3)$

Along with the initialization list **list**, 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 *(stclim)* 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 *(objerr)* r Default $= \epsilon^{\frac{1}{4}}$

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 *(objsfg)* r Default $= \epsilon^{\frac{1}{2}}$

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 *(objval)* r

This parameter may be set if you wish e05jb 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.
