NAG Library Function Document

nag_opt_bounds_qa_no_deriv (e04jcc)

1 Purpose

nag_opt_bounds_qa_no_deriv (e04jcc) is an easy-to-use algorithm that uses methods of quadratic approximation to find a minimum of an objective function F over $\mathbf{x} \in R^n$, subject to fixed lower and upper bounds on the independent variables $x_1, x_2, \ldots, x_n$. Derivatives of F are not required.

The function is intended for functions that are continuous and that have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities). Efficiency is maintained for large n.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_bounds_qa_no_deriv (
     void (*objfun)(Integer n, const double x[], double *f, Nag_Comm *comm,
          Integer *inform),
     Integer n, Integer npt, double x[], const double bl[],
     const double bu[], double rhobeg, double rhoend,
     void (*monfun)(Integer n, Integer nf, const double x[], double f,
          double rho, Nag_Comm *comm, Integer *inform),
     Integer maxcal, double *f, Integer *nf, Nag_Comm *comm, NagError *fail)
```

3 Description

nag_opt_bounds_qa_no_deriv (e04jcc) is applicable to problems of the form:

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

where F is a nonlinear scalar function whose derivatives may be unavailable, and where the bound vectors are elements of R^n. Relational operators between vectors are interpreted elementwise.

Fixing variables (that is, setting $\ell_i = u_i$ for some i) is allowed in nag_opt_bounds_qa_no_deriv (e04jcc).

You must supply a function to calculate the value of F at any given point $\mathbf{x}$.

The method used by nag_opt_bounds_qa_no_deriv (e04jcc) is based on BOBYQA, the method of Bound Optimization BY Quadratic Approximation described in Powell (2009). In particular, each iteration of nag_opt_bounds_qa_no_deriv (e04jcc) generates a quadratic approximation Q to F that agrees with F at m automatically chosen interpolation points. The value of m is a constant prescribed by you. Updates to the independent variables mostly occur from approximate solutions to trust-region subproblems, using the current quadratic model.

4 References

Powell M J D (2009) The BOBYQA algorithm for bound constrained optimization without derivatives *Report DAMTP 2009/NA06* University of Cambridge http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf

5 Arguments

1: **objfun** – function, supplied by the user *External Function*

objfun must evaluate the objective function F at a specified vector $\mathbf{x}$.

The specification of **objfun** is:

```
void objfun (Integer n, const double x[], double *f, Nag_Comm *comm,
     Integer *inform)
```

1: **n** – Integer *Input*

On entry: n, the number of independent variables.

2: **x**[**n**] – const double *Input*

On entry: $\mathbf{x}$, the vector at which the objective function is to be evaluated.

3: **f** – double * *Output*

On exit: must be set to the value of the objective function at $\mathbf{x}$, unless you have specified termination of the current problem using **inform**.

4: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **objfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_bounds_qa_no_deriv (e04jcc) you may allocate memory and initialize these pointers with various quantities for use by **objfun** when called from nag_opt_bounds_qa_no_deriv (e04jcc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

5: **inform** – Integer * *Output*

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

2: **n** – Integer *Input*

On entry: n, the number of independent variables.

Constraint: $\mathbf{n} \geq 2$ and $n_r \geq 2$, where n_r denotes the number of non-fixed variables.

3: **npt** – Integer *Input*

On entry: m, the number of interpolation conditions imposed on the quadratic approximation at each iteration.

Suggested value: $\mathbf{npt} = 2 \times n_r + 1$, where n_r denotes the number of non-fixed variables.

Constraint: $n_r + 2 \leq \mathbf{npt} \leq \frac{(n_r+1)\times(n_r+2)}{2}$, where n_r denotes the number of non-fixed variables.

4: **x**[**n**] – double *Input/Output*

On entry: an estimate of the position of the minimum. If any component is out-of-bounds it is replaced internally by the bound it violates.

On exit: the lowest point found during the calculations. Thus, if **fail.code** = NE_NOERROR on exit, **x** is the position of the minimum.

5: **bl**[**n**] – const double *Input*

6: **bu**[**n**] – const double *Input*

On entry: the fixed vectors of bounds: the lower bounds $\boldsymbol{\ell}$ and the upper bounds **u**, respectively. To signify that a variable is unbounded you should choose a large scalar r appropriate to your problem, then set the lower bound on that variable to $-r$ and the upper bound to r. For well-scaled problems $r = r_{\max}^{\frac{1}{4}}$ may be suitable, where $r_{\max}$ denotes the largest positive model number (see nag_real_largest_number (X02ALC)).

Constraints:

if $\mathbf{x}[i-1]$ is to be fixed at $\mathbf{bl}[i-1]$, then $\mathbf{bl}[i-1] = \mathbf{bu}[i-1]$;
otherwise $\mathbf{bu}[i-1] - \mathbf{bl}[i-1] \geq 2.0 \times \mathbf{rhobeg}$, for $i = 1, 2, \ldots, \mathbf{n}$.

7: **rhobeg** – double *Input*

On entry: an initial lower bound on the value of the trust-region radius.

Suggested value: **rhobeg** should be about one tenth of the greatest expected overall change to a variable: the initial quadratic model will be constructed by taking steps from the initial **x** of length **rhobeg** along each coordinate direction.

Constraints:

rhobeg > 0.0;
rhobeg $\geq$ **rhoend**.

8: **rhoend** – double *Input*

On entry: a final lower bound on the value of the trust-region radius.

Suggested value: **rhoend** should indicate the absolute accuracy that is required in the final values of the variables.

Constraint: **rhoend** $\geq$ $macheps$, where $macheps$ = nag_machine_precision, the ***machine precision***..

9: **monfun** – function, supplied by the user *External Function*

monfun may be used to monitor the optimization process. It is invoked every time a new trust-region radius is chosen.

If no monitoring is required, **monfun** may be specified as NULLFN.

The specification of **monfun** is:

```
void monfun (Integer n, Integer nf, const double x[], double f,
     double rho, Nag_Comm *comm, Integer *inform)
```

1: **n** – Integer *Input*

On entry: n, the number of independent variables.

2: **nf** – Integer *Input*

On entry: the cumulative number of calls made to **objfun**.

3: **x**[**n**] – const double *Input*

On entry: the current best point.

4: **f** – double *Input*

On entry: the value of **objfun** at **x**.

5: **rho** – double *Input*

On entry: a lower bound on the current trust-region radius.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **monfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_bounds_qa_no_deriv (e04jcc) you may allocate memory and initialize these pointers with various quantities for use by **monfun** when called from nag_opt_bounds_qa_no_deriv (e04jcc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7: **inform** – Integer * *Output*

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

10: **maxcal** – Integer *Input*

On entry: the maximum permitted number of calls to **objfun**.

Constraint: **maxcal** ≥ 1.

11: **f** – double * *Output*

On exit: the function value at the lowest point found (**x**).

12: **nf** – Integer * *Output*

On exit: unless **fail.code** = NE_RESCUE_FAILED, NE_TOO_MANY_FEVALS, NE_TR_STEP_FAILED or NE_USER_STOP on exit, the total number of calls made to **objfun**.

13: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

14: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

nag_opt_bounds_qa_no_deriv (e04jcc) returns with **fail.code** = NE_NOERROR if the final trust-region radius has reached its lower bound **rhoend**.

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_BOUND

On entry, **rhobeg** $= \langle value \rangle$, **bl**$[i-1] = \langle value \rangle$, **bu**$[i-1] = \langle value \rangle$ and $i = \langle value \rangle$.
Constraint: if **bl**$[i-1] \neq$ **bu**$[i-1]$ in coordinate i, then **bu**$[i-1] -$ **bl**$[i-1] \geq 2 \times$ **rhobeg**.

NE_INT

On entry, **maxcal** $= \langle value \rangle$.
Constraint: **maxcal** ≥ 1.

There were $n_r = \langle value \rangle$ unequal bounds.
Constraint: $n_r \geq 2$.

There were $n_r = \langle value \rangle$ unequal bounds and **npt** $= \langle value \rangle$ on entry.
Constraint: $n_r + 2 \leq$ **npt** $\leq \frac{(n_r+1)\times(n_r+2)}{2}$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL

On entry, **rhobeg** $= \langle value \rangle$.
Constraint: **rhobeg** > 0.0.

On entry, **rhoend** $= \langle value \rangle$.
Constraint: **rhoend** $\geq$ *macheps*, where *macheps* = nag_machine_precision, the ***machine precision***.

NE_REAL_2

On entry, **rhobeg** $= \langle value \rangle$ and **rhoend** $= \langle value \rangle$.
Constraint: **rhoend** $\leq$ **rhobeg**.

NE_RESCUE_FAILED

A rescue procedure has been called in order to correct damage from rounding errors when computing an update to a quadratic approximation of F, but no further progess could be made. Check your specification of **objfun** and whether the function needs rescaling. Try a different initial **x**.

NE_TOO_MANY_FEVALS

The function evaluations limit was reached: **objfun** has been called **maxcal** times.

NE_TR_STEP_FAILED

The predicted reduction in a trust-region step was non-positive. Check your specification of **objfun** and whether the function needs rescaling. Try a different initial **x**.

NE_USER_STOP

User-supplied monitoring function requested termination.

User-supplied objective function requested termination.

7 Accuracy

Experience shows that, in many cases, on successful termination the ∞-norm distance from the best point **x** to a local minimum of F is less than $10 \times$ **rhoend**, unless **rhoend** is so small that such accuracy is unattainable.

8 Parallelism and Performance

nag_opt_bounds_qa_no_deriv (e04jcc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

For each invocation of nag_opt_bounds_qa_no_deriv (e04jcc), local workspace arrays of fixed length are allocated internally. The total size of these arrays amounts to $(\mathbf{npt}+6)\times(\mathbf{npt}+n_r)+\frac{n_r\times(3n_r+21)}{2}$ double elements and n_r Integer elements, where n_r denotes the number of non-fixed variables; that is, the total size is $\mathcal{O}(n_r^4)$. If you follow the recommendation for the choice of **npt** on entry, this total size reduces to $\mathcal{O}(n_r^2)$.

Usually the total number of function evaluations (**nf**) is substantially less than $\mathcal{O}(n_r^2)$, and often, if $\mathbf{npt} = 2 \times n_r + 1$ on entry, **nf** is only of magnitude n_r or less.

10 Example

This example involves the minimization of

$$F = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

subject to

$$\begin{array}{r} 1 \le x_1 \le 3 \\ -2 \le x_2 \le 0 \\ 1 \le x_4 \le 3, \end{array}$$

starting from the initial guess $(3, -1, 0, 1)$.

10.1 Program Text

```
/* nag_opt_bounds_qa_no_deriv (e04jcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
```

```
 */

#include <nag.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL objfun(Integer n, const double x[], double *f,
                              Nag_Comm *comm, Integer *inform);
  static void NAG_CALL monfun(Integer n, Integer nf, const double x[],
                              double f, double rho, Nag_Comm *comm,
                              Integer *inform);
#ifdef __cplusplus
}
#endif

int main(void)
{
  static double ruser[2] = { -1.0, -1.0 };
  Integer exit_status = 0;
  double rhobeg, rhoend, f;
  Integer i, n, nf, npt, maxcal;
  double *bl = 0, *bu = 0, *x = 0;
  NagError fail;
  Nag_Comm comm;

  INIT_FAIL(fail);

  printf("nag_opt_bounds_qa_no_deriv (e04jcc) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  maxcal = 500;
  rhobeg = 1.0e-1;
  rhoend = 1.0e-6;
  n = 4;
  npt = 2 * n + 1;

  if (!(x = NAG_ALLOC(n, double)) ||
      !(bl = NAG_ALLOC(n, double)) || !(bu = NAG_ALLOC(n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Set bounds on variables */
  /* x[2] is not bounded, so we set bl[2] to a large negative
   * number and bu[2] to a large positive number
   */
  bl[0] = 1.0;
  bl[1] = -2.0;
  bl[2] = -1.0e10;
  bl[3] = 1.0;
  bu[0] = 3.0;
  bu[1] = 0.0;
  bu[2] = 1.0e10;
  bu[3] = 3.0;
  x[0] = 3.0;
  x[1] = -1.0;
  x[2] = 0.0;
  x[3] = 1.0;
```

```
  /* Call optimization routine */
  /* nag_opt_bounds_qa_no_deriv (e04jcc).
     Bound-constrained optimization by quadratic approximations. */
  nag_opt_bounds_qa_no_deriv(objfun, n, npt, x, bl, bu, rhobeg, rhoend,
                             monfun, maxcal, &f, &nf, &comm, &fail);

  if (fail.code == NE_NOERROR ||
      fail.code == NE_TOO_MANY_FEVALS ||
      fail.code == NE_TR_STEP_FAILED ||
      fail.code == NE_RESCUE_FAILED || fail.code == NE_USER_STOP) {

    if (fail.code == NE_NOERROR) {
      printf("Successful exit.\n");
    }

    printf("Function value at lowest point found is %11.3f\n", f);
    printf("The corresponding x is:");
    for (i = 0; i <= n - 1; ++i) {
      printf(" %11.3f", x[i]);
    }
    printf("\n");
  }
  else {
    exit_status = 1;
  }

  if (fail.code != NE_NOERROR) {
    printf("%s\n", fail.message);
  }

END:
  NAG_FREE(x);
  NAG_FREE(bl);
  NAG_FREE(bu);

  return exit_status;
}

static void NAG_CALL objfun(Integer n, const double x[], double *f,
                            Nag_Comm *comm, Integer *inform)
{
  /* Routine to evaluate objective function. */

  double a, b, c, d, x1, x2, x3, x4;

  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback objfun, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  *inform = 0;
  x1 = x[0];
  x2 = x[1];
  x3 = x[2];
  x4 = x[3];

  /* Supply a single function value */
  a = x1 + 10.0 * x2;
  b = x3 - x4;
  c = x2 - 2.0 * x3, c *= c;
  d = x1 - x4, d *= d;
  *f = a * a + 5.0 * b * b + c * c + 10.0 * d * d;
}

static void NAG_CALL monfun(Integer n, Integer nf, const double x[], double f,
                            double rho, Nag_Comm *comm, Integer *inform)
{
  /* Monitoring routine */
  Integer j;
  Nag_Boolean verbose;

  if (comm->user[1] == -1.0) {
```

```
    printf("(User-supplied callback monfun, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  *inform = 0;

  printf("\nMonitoring: new trust region radius = %13.3e\n", rho);
  verbose = Nag_FALSE; /* Set this to Nag_TRUE to get more detailed output */
  if (verbose)
    {
      printf("Number of function evaluations = %16" NAG_IFMT "\n", nf);
      printf("Current function value = %13.5f\n", f);
      printf("The corresponding x is:\n");
      for (j = 0; j <= n - 1; ++j) {
        printf(" %13.5e", x[j]);
      }
      printf("\n");
    }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_opt_bounds_qa_no_deriv (e04jcc) Example Program Results
(User-supplied callback objfun, first invocation.)
(User-supplied callback monfun, first invocation.)

Monitoring: new trust region radius =     1.000e-02

Monitoring: new trust region radius =     1.000e-03

Monitoring: new trust region radius =     1.000e-04

Monitoring: new trust region radius =     1.000e-05

Monitoring: new trust region radius =     1.000e-06
Successful exit.
Function value at lowest point found is       2.434
The corresponding x is:       1.000      -0.085       0.409       1.000
```