NAG Library Function Document

nag_opt_simplex_easy (e04cbc)

1 Purpose

nag_opt_simplex_easy (e04cbc) minimizes a general function $F(\mathbf{x})$ of n independent variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ by the Nelder and Mead simplex method (see Nelder and Mead (1965)). Derivatives of the function need not be supplied.

2 Specification

```
#include <nag.h>
#include <nage04.h>

void nag_opt_simplex_easy (Integer n, double x[], double *f, double tolf,
     double tolx,
     void (*funct)(Integer n, const double xc[], double *fc, Nag_Comm *comm),
     void (*monit)(double fmin, double fmax, const double sim[], Integer n,
          Integer ncall, double serror, double vratio, Nag_Comm *comm),
     Integer maxcal, Nag_Comm *comm, NagError *fail)
```

3 Description

nag_opt_simplex_easy (e04cbc) finds an approximation to a minimum of a function F of n variables. You must supply a function to calculate the value of F for any set of values of the variables.

The method is iterative. A simplex of $n + 1$ points is set up in the n-dimensional space of the variables (for example, in 2 dimensions the simplex is a triangle) under the assumption that the problem has been scaled so that the values of the independent variables at the minimum are of order unity. The starting point you have provided is the first vertex of the simplex, the remaining n vertices are generated by nag_opt_simplex_easy (e04cbc). The vertex of the simplex with the largest function value is reflected in the centre of gravity of the remaining vertices and the function value at this new point is compared with the remaining function values. Depending on the outcome of this test the new point is accepted or rejected, a further expansion move may be made, or a contraction may be carried out. See Nelder and Mead (1965) and Parkinson and Hutchinson (1972) for more details. When no further progress can be made the sides of the simplex are reduced in length and the method is repeated.

The method can be slow, but computational bottlenecks have been reduced following Singer and Singer (2004). However, nag_opt_simplex_easy (e04cbc) is robust, and therefore very useful for functions that are subject to inaccuracies.

There are the following options for successful termination of the method: based only on the function values at the vertices of the current simplex (see (1)); based only on a volume ratio between the current simplex and the initial one (see (2)); or based on which one of the previous two tests passes first. The volume test may be useful if F is discontinuous, while the function-value test should be sufficient on its own if F is continuous.

4 References

Nelder J A and Mead R (1965) A simplex method for function minimization *Comput. J.* **7** 308–313

Parkinson J M and Hutchinson D (1972) An investigation into the efficiency of variants of the simplex method *Numerical Methods for Nonlinear Optimization* (ed F A Lootsma) Academic Press

Singer S and Singer S (2004) Efficient implementation of the Nelder–Mead search algorithm *Appl. Num. Anal. Comp. Math.* **1(3)** 524–534

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of variables.

Constraint: $\mathbf{n} \geq 1$.

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

On entry: a guess at the position of the minimum. Note that the problem should be scaled so that the values of the $\mathbf{x}[i-1]$ are of order unity.

On exit: the value of **x** corresponding to the function value in **f**.

3: **f** – double * *Output*

On exit: the lowest function value found.

4: **tolf** – double *Input*

On entry: the error tolerable in the function values, in the following sense. If f_i, for $i = 1, 2, \ldots, n+1$, are the individual function values at the vertices of the current simplex, and if f_m is the mean of these values, then you can request that nag_opt_simplex_easy (e04cbc) should terminate if

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}(f_i - f_m)^2} < \mathbf{tolf}. \quad (1)$$

You may specify $\mathbf{tolf} = 0$ if you wish to use only the termination criterion (2) on the spatial values: see the description of **tolx**.

Constraint: **tolf** must be greater than or equal to the ***machine precision*** (see Chapter x02), or if **tolf** equals zero then **tolx** must be greater than or equal to the ***machine precision***.

5: **tolx** – double *Input*

On entry: the error tolerable in the spatial values, in the following sense. If LV denotes the 'linearized' volume of the current simplex, and if LV_{init} denotes the 'linearized' volume of the initial simplex, then you can request that nag_opt_simplex_easy (e04cbc) should terminate if

$$\frac{LV}{LV_{\text{init}}} < \mathbf{tolx}. \quad (2)$$

You may specify $\mathbf{tolx} = 0$ if you wish to use only the termination criterion (1) on function values: see the description of **tolf**.

Constraint: **tolx** must be greater than or equal to the ***machine precision*** (see Chapter x02), or if **tolx** equals zero then **tolf** must be greater than or equal to the ***machine precision***.

6: **funct** – function, supplied by the user *External Function*

funct must evaluate the function F at a specified point. It should be tested separately before being used in conjunction with nag_opt_simplex_easy (e04cbc).

The specification of **funct** is:

```
void funct (Integer n, const double xc[], double *fc, Nag_Comm *comm)
```

1: **n** – Integer *Input*

On entry: n, the number of variables.

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

On entry: the point at which the function value is required.

3: **fc** – double * *Output*

On exit: the value of the function F at the current point **x**.

4: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_simplex_easy (e04cbc) you may allocate memory and initialize these pointers with various quantities for use by **funct** when called from nag_opt_simplex_easy (e04cbc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7: **monit** – function, supplied by the user *External Function*

monit may be used to monitor the optimization process. It is invoked once every iteration.

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

The specification of **monit** is:

```
void monit (double fmin, double fmax, const double sim[], Integer n,
     Integer ncall, double serror, double vratio, Nag_Comm *comm)
```

1: **fmin** – double *Input*

On entry: the smallest function value in the current simplex.

2: **fmax** – double *Input*

On entry: the largest function value in the current simplex.

3: **sim**[$(\mathbf{n}+\mathbf{1})\times\mathbf{n}$] – const double *Input*

On entry: the $n+1$ position vectors of the current simplex, where **sim**[$(j-1)\times(n+1)+i-1$] is the jth coordinate of the ith position vector.

4: **n** – Integer *Input*

On entry: n, the number of variables.

5: **ncall** – Integer *Input*

On entry: the number of times that **funct** has been called so far.

6: **serror** – double *Input*

On entry: the current value of the standard deviation in function values used in termination test (1).

7: **vratio** – double *Input*

On entry: the current value of the linearized volume ratio used in termination test (2).

8: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_simplex_easy (e04cbc) you may allocate memory and initialize these pointers with various quantities for use by **monit** when called from nag_opt_simplex_easy (e04cbc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

8: **maxcal** – Integer *Input*

On entry: the maximum number of function evaluations to be allowed.

Constraint: **maxcal** ≥ 1.

9: **comm** – Nag_Comm *

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

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

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

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_INT

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

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

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, **tolf** $= 0.0$ and **tolx** $= \langle value \rangle$.
Constraint: if **tolf** $= 0.0$ then **tolx** is greater than or equal to the ***machine precision***.

On entry, **tolx** $= 0.0$ and **tolf** $= \langle value \rangle$.
Constraint: if **tolx** $= 0.0$ then **tolf** is greater than or equal to the ***machine precision***.

NE_REAL_2

On entry, **tolf** $= \langle value \rangle$ and **tolx** $= \langle value \rangle$.
Constraint: if **tolf** $\neq 0.0$ and **tolx** $\neq 0.0$ then both should be greater than or equal to the ***machine precision***.

NW_TOO_MANY_FEVALS

maxcal function evaluations have been completed without any other termination test passing. Check the coding of **funct** before increasing the value of **maxcal**.

7 Accuracy

On a successful exit the accuracy will be as defined by **tolf** or **tolx**, depending on which criterion was satisfied first.

8 Parallelism and Performance

nag_opt_simplex_easy (e04cbc) is not threaded in any implementation.

9 Further Comments

Local workspace arrays of fixed lengths (depending on **n**) are allocated internally by nag_opt_simplex_easy (e04cbc). The total size of these arrays amounts to $\mathbf{n}^2 + 6\mathbf{n} + 2$ double elements.

The time taken by nag_opt_simplex_easy (e04cbc) depends on the number of variables, the behaviour of the function and the distance of the starting point from the minimum. Each iteration consists of 1 or 2 function evaluations unless the size of the simplex is reduced, in which case $n + 1$ function evaluations are required.

10 Example

This example finds a minimum of the function

$$F(x_1, x_2) = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right).$$

This example uses the initial point $(-1, 1)$ (see Section 10.3), and we expect to reach the minimum at $(0.5, -1)$.

10.1 Program Text

```
/* nag_opt_simplex_easy (e04cbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <math.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx02.h>

#ifdef __cplusplus
extern "C"
{
```

```
#endif
  static void NAG_CALL funct(const Integer n, const double *xc, double *fc,
                             Nag_Comm *comm);
  static void NAG_CALL monit(const double fmin, const double fmax,
                             const double sim[], const Integer n,
                             const Integer ncall, const double serror,
                             const double vratio, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  double f, tolf, tolx;
  Integer exit_status, i, monitoring, maxcal = 100, n = 2;
  NagError fail;

  /* Arrays */
  static double ruser[2] = { -1.0, -1.0 };
  double *x = 0;

  Nag_Comm comm;

  exit_status = 0;

  INIT_FAIL(fail);

  printf("nag_opt_simplex_easy (e04cbc) Example Program Results\n");

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

  /* Allocate memory */
  if (!(x = NAG_ALLOC(n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Set monitoring to a nonzero value to obtain monitoring information */
  monitoring = 0;
  comm.p = (Pointer) &monitoring;

  /* Starting values */
  x[0] = -1.0;
  x[1] = 1.0;
  tolf = sqrt(nag_machine_precision);
  tolx = sqrt(tolf);

  nag_opt_simplex_easy(n, x, &f, tolf, tolx, funct, monit, maxcal, &comm,
                       &fail);

  if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_simplex_easy (e04cbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("The final function value is %12.4f\n", f);
  printf("at the point");
  for (i = 1; i <= n; ++i) {
    printf(" %12.4f", x[i - 1]);
  }
  printf("\n");

END:
  NAG_FREE(x);

  return exit_status;
```

```
}

static void NAG_CALL funct(const Integer n, const double *xc, double *fc,
                           Nag_Comm *comm)
{
  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback funct, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  *fc = exp(xc[0]) * (4.0 * xc[0] * (xc[0] + xc[1]) +
                      2.0 * xc[1] * (xc[1] + 1.0) + 1.0);
}

static void NAG_CALL monit(const double fmin, const double fmax,
                           const double sim[], const Integer n,
                           const Integer ncall, const double serror,
                           const double vratio, Nag_Comm *comm)
{
#define SIM(I, J) sim[(J-1)*(n+1) + (I-1)]

  Integer i, j;
  Integer monitoring = *(Integer *) comm->p;

  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback monit, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  if (monitoring != 0) {
    printf("\nThere have been %5" NAG_IFMT " function calls\n", ncall);
    printf("The smallest function value is %10.4f\n", fmin);
    printf("\nThe simplex is\n");
    for (i = 1; i <= n + 1; ++i) {
      for (j = 1; j <= n; ++j) {
        printf(" %13.4e", SIM(i, j));
      }
      printf("\n");
    }
    printf("\nThe standard deviation in function values at the"
           " vertices of the simplex is %10.4f\n", serror);
    printf("The linearized volume ratio of the current simplex"
           " to the starting one is %10.4f\n", vratio);
  }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_opt_simplex_easy (e04cbc) Example Program Results
(User-supplied callback funct, first invocation.)
(User-supplied callback monit, first invocation.)
The final function value is       0.0000
at the point       0.5000      -0.9999
```

Example Program
Contours of *F* Showing the Initial Point (X) and Local Minimum (*)