NAG Toolbox

nag_opt_bounds_mod_deriv_easy (e04kz)

1 Purpose

nag_opt_bounds_mod_deriv_easy (e04kz) is an easy-to-use modified Newton algorithm for finding a minimum of a function $F(x_1, x_2, \ldots, x_n)$, subject to fixed upper and lower bounds on the independent variables $x_1, x_2, \ldots, x_n$, when first derivatives of F are available. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Syntax

```
[bl, bu, x, f, g, user, ifail] = nag_opt_bounds_mod_deriv_easy(ibound, funct2, bl, bu, x, 'n', n, 'user', user)
[bl, bu, x, f, g, user, ifail] = e04kz(ibound, funct2, bl, bu, x, 'n', n, 'user', user)
```

3 Description

nag_opt_bounds_mod_deriv_easy (e04kz) is applicable to problems of the form:

$$\text{Minimize } F(x_1, x_2, \ldots, x_n) \quad \text{subject to} \quad l_j \leq x_j \leq u_j, \quad j = 1, 2, \ldots, n$$

when first derivatives are known.

Special provision is made for problems which actually have no bounds on the x_j, problems which have only non-negativity bounds, and problems in which $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$. You must supply a function to calculate the values of $F(x)$ and its first derivatives at any point x.

From a starting point you supplied there is generated, on the basis of estimates of the gradient of the curvature of $F(x)$, a sequence of feasible points which is intended to converge to a local minimum of the constrained function.

4 References

Gill P E and Murray W (1976) Minimization subject to bounds on the variables *NPL Report NAC 72* National Physical Laboratory

5 Parameters

5.1 Compulsory Input Parameters

1: **ibound** – INTEGER

Indicates whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:

ibound $= 0$
If you are supplying all the l_j and u_j individually.

ibound $= 1$
If there are no bounds on any x_j.

ibound $= 2$
If all the bounds are of the form $0 \leq x_j$.

ibound $= 3$
If $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.

Constraint: $0 \leq$ **ibound** ≤ 3.

2: **funct2** – SUBROUTINE, supplied by the user.

You must supply this function to calculate the values of the function $F(x)$ and its first derivatives $\dfrac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with nag_opt_bounds_mod_deriv_easy (e04kz) (see Chapter E04).

```
[fc, gc, user] = funct2(n, xc, user)
```

Input Parameters

1: **n** – INTEGER

The number n of variables.

2: **xc**(**n**) – REAL (KIND=nag_wp) array

The point x at which the function and derivatives are required.

3: **user** – INTEGER array

funct2 is called from nag_opt_bounds_mod_deriv_easy (e04kz) with the object supplied to nag_opt_bounds_mod_deriv_easy (e04kz).

Output Parameters

1: **fc** – REAL (KIND=nag_wp)

The value of the function F at the current point x,

2: **gc**(**n**) – REAL (KIND=nag_wp) array

gc(j) must be set to the value of the first derivative $\dfrac{\partial F}{\partial x_j}$ at the point x, for $j = 1, 2, \ldots, n$.

3: **user** – INTEGER array

3: **bl**(**n**) – REAL (KIND=nag_wp) array

The lower bounds l_j.

If **ibound** is set to 0, you must set **bl**(j) to l_j, for $j = 1, 2, \ldots, n$. (If a lower bound is not specified for a particular x_j, the corresponding **bl**(j) should be set to -10^6.)

If **ibound** is set to 3, you must set **bl**(1) to l_1; nag_opt_bounds_mod_deriv_easy (e04kz) will then set the remaining elements of **bl** equal to **bl**(1).

4: **bu**(**n**) – REAL (KIND=nag_wp) array

The upper bounds u_j.

If **ibound** is set to 0, you must set **bu**(j) to u_j, for $j = 1, 2, \ldots, n$. (If an upper bound is not specified for a particular x_j, the corresponding **bu**(j) should be set to 10^6.)

If **ibound** is set to 3, you must set **bu**(1) to u_1; nag_opt_bounds_mod_deriv_easy (e04kz) will then set the remaining elements of **bu** equal to **bu**(1).

5: **x**(**n**) – REAL (KIND=nag_wp) array

x(j) must be set to a guess at the jth component of the position of the minimum, for $j = 1, 2, \ldots, n$. The function checks the gradient at the starting point, and is more likely to detect any error in your programming if the initial **x**(j) are nonzero and mutually distinct.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **bl**, **bu**, **x**. (An error is raised if these dimensions are not equal.)

The number n of independent variables.

Constraint: **n** ≥ 1.

2: **user** – INTEGER array

user is not used by nag_opt_bounds_mod_deriv_easy (e04kz), but is passed to **funct2**. 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 Output Parameters

1: **bl**(**n**) – REAL (KIND=nag_wp) array

The lower bounds actually used by nag_opt_bounds_mod_deriv_easy (e04kz).

2: **bu**(**n**) – REAL (KIND=nag_wp) array

The upper bounds actually used by nag_opt_bounds_mod_deriv_easy (e04kz).

3: **x**(**n**) – REAL (KIND=nag_wp) array

The lowest point found during the calculations of the position of the minimum.

4: **f** – REAL (KIND=nag_wp)

The value of $F(x)$ corresponding to the final point stored in **x**.

5: **g**(**n**) – REAL (KIND=nag_wp) array

The value of $\dfrac{\partial F}{\partial x_j}$ corresponding to the final point stored in **x**, for $j = 1, 2, \ldots, n$; the value of **g**(j) for variables not on a bound should normally be close to zero.

6: **user** – INTEGER array

7: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: nag_opt_bounds_mod_deriv_easy (e04kz) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$

On entry, **n** < 1,
or **ibound** < 0,

or **ibound** > 3,
or **ibound** $= 0$ and **bl**$(j) >$ **bu**(j) for some j,
or **ibound** $= 3$ and **bl**$(1) >$ **bu**(1),
or $liw <$ **n** $+ 2$,
or $lw < \max(10,$ **n** $\times$ (**n** $+ 7))$.

ifail $= 2$

There has been a large number of function evaluations, yet the algorithm does not seem to be converging. The calculations can be restarted from the final point held in **x**. The error may also indicate that $F(x)$ has no minimum.

ifail $= 3$ (*warning*)

The conditions for a minimum have not all been met but a lower point could not be found and the algorithm has failed.

ifail $= 4$

Not used. (This value of the argument is included to make the significance of **ifail** $= 5$ etc. consistent in the easy-to-use functions.)

ifail $= 5$ (*warning*)
ifail $= 6$ (*warning*)
ifail $= 7$ (*warning*)
ifail $= 8$ (*warning*)

There is some doubt about whether the point x found by nag_opt_bounds_mod_deriv_easy (e04kz) is a minimum. The degree of confidence in the result decreases as **ifail** increases. Thus, when **ifail** $= 5$ it is probable that the final x gives a good estimate of the position of a minimum, but when **ifail** $= 8$ it is very unlikely that the function has found a minimum.

ifail $= 9$

In the search for a minimum, the modulus of one of the variables has become very large $(\sim 10^6)$. This indicates that there is a mistake in **funct2**, that your problem has no finite solution, or that the problem needs rescaling (see Section 9).

ifail $= 10$

It is very likely that you have made an error in forming the gradient.

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

If you are dissatisfied with the result (e.g., because **ifail** $= 5$, 6, 7 or 8), it is worth restarting the calculations from a different starting point (not the point at which the failure occurred) in order to avoid the region which caused the failure. If persistent trouble occurs and it is possible to calculate second derivatives it may be advisable to change to a function which uses second derivatives (see the E04 Chapter Introduction).

7 Accuracy

When a successful exit is made then, for a computer with a mantissa of t decimals, one would expect to get about $t/2 - 1$ decimals accuracy in x and about $t - 1$ decimals accuracy in F, provided the problem is reasonably well scaled.

8 Further Comments

The number of iterations required depends on the number of variables, the behaviour of $F(x)$ and the distance of the starting point from the solution. The number of operations performed in an iteration of nag_opt_bounds_mod_deriv_easy (e04kz) is roughly proportional to $n^3 + O(n^2)$. In addition, each iteration makes at least $m + 1$ calls of **funct2** where m is the number of variables not fixed on bounds. So unless $F(x)$ and the gradient vector can be evaluated very quickly, the run time will be dominated by the time spent in **funct2**.

Ideally the problem should be scaled so that at the solution the value of $F(x)$ and the corresponding values of $x_1, x_2, \ldots, x_n$ are in the range $(-1, +1)$, and so that at points a unit distance away from the solution, F is approximately a unit value greater than at the minimum. It is unlikely that you will be able to follow these recommendations very closely, but it is worth trying (by guesswork), as sensible scaling will reduce the difficulty of the minimization problem, so that nag_opt_bounds_mod_deriv_easy (e04kz) will take less computer time.

9 Example

A program to minimize

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

subject to

$$\begin{array}{rcccl} 1 & \leq & x_1 & \leq & 3 \\ -2 & \leq & x_2 & \leq & 0 \\ 1 & \leq & x_4 & \leq & 3 \end{array}$$

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

In practice, it is worth trying to make **funct2** as efficient as possible. This has not been done in the example program for reasons of clarity.

9.1 Program Text

```
function e04kz_example

fprintf('e04kz example results\n\n');

ibound = nag_int(0);
bl = [ 1; -2; -1000000; 1];
bu = [ 3;  0;  1000000; 3];
x  = [ 3; -1;        0; 1];

% Catch warnings and assume ifail=3,5 gives a good estimate
wstat = warning();
warning('OFF');
[bl, bu, x, f, g, user, ifail] = e04kz(ibound, @funct2, bl, bu, x);
if (ifail == 0 || ifail == 5 | ifail == 3)
  fprintf('\nMinimum found at x: ');
  fprintf(' %9.4f',x);
  fprintf('\nGradients at x,  g: ');
  fprintf(' %9.4f',g);
  fprintf('\nMinimum value     :  %9.4f\n\n',f);
else
  fprintf('\n Error: e04kz returns ifail = %d\n',ifail);
end
warning(wstat);
```

```
function [fc, gc, user] = funct2(n, xc, user)
  gc = zeros(n, 1);
  fc = 0;
  x1 = xc(1) + 10*xc(2);
  x2 = xc(3) -    xc(4);
  x3 = xc(2) -  2*xc(3);
  x4 = xc(1) -    xc(4);
  fc = x1^2 + 5*x2^2 + x3^4 + 10*x4^4;
  gc(1) =   2*x1 + 40*x4^3;
  gc(2) =  20*x1 +  4*x3^3;
  gc(3) =  10*x2 -  8*x3^3;
  gc(4) = -10*x2 - 40*x4^3;
```

9.2 Program Results

```
     e04kz example results

Minimum found at x:     1.0000   -0.0852    0.4093    1.0000
Gradients at x,  g:     0.2953   -0.0000    0.0000    5.9070
Minimum value     :     2.4338
```