NAG Toolbox

nag_opt_bounds_quasi_deriv_easy (e04ky)

1 Purpose

nag_opt_bounds_quasi_deriv_easy (e04ky) is an easy-to-use quasi-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, iw, w, user, ifail] = nag_opt_bounds_quasi_deriv_easy(ibound, funct2, bl, bu, x, 'n', n, 'liw', liw, 'lw', lw, 'user', user)
[bl, bu, x, f, g, iw, w, user, ifail] = e04ky(ibound, funct2, bl, bu, x, 'n', n, 'liw', liw, 'lw', lw, 'user', user)
```

3 Description

nag_opt_bounds_quasi_deriv_easy (e04ky) 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 available.

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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$. 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 curvature of $F(x)$, a sequence of feasible points which is intended to converge to a local minimum of the constrained function. An attempt is made to verify that the final point is a minimum.

A typical iteration starts at the current point x where n_z (say) variables are free from both their bounds. The projected gradient vector g_z, whose elements are the derivatives of $F(x)$ with respect to the free variables, is known. A unit lower triangular matrix L and a diagonal matrix D (both of dimension n_z), such that LDL^{T} is a positive definite approximation of the matrix of second derivatives with respect to the free variables (i.e., the projected Hessian) are also held. The equations

$$LDL^{\mathrm{T}}p_z = -g_z$$

are solved to give a search direction p_z, which is expanded to an n-vector p by an insertion of appropriate zero elements. Then α is found such that $F(x + \alpha p)$ is approximately a minimum (subject to the fixed bounds) with respect to α; x is replaced by $x + \alpha p$, and the matrices L and D are updated so as to be consistent with the change produced in the gradient by the step αp. If any variable actually reaches a bound during the search along p, it is fixed and n_z is reduced for the next iteration.

There are two sets of convergence criteria – a weaker and a stronger. Whenever the weaker criteria are satisfied, the Lagrange multipliers are estimated for all the active constraints. If any Lagrange multiplier estimate is significantly negative, then one of the variables associated with a negative Lagrange multiplier estimate is released from its bound and the next search direction is computed in the extended subspace (i.e., n_z is increased). Otherwise minimization continues in the current subspace provided that this is practicable. When it is not, or when the stronger convergence criteria are already satisfied, then, if one or more Lagrange multiplier estimates are close to zero, a slight perturbation is made in the values of the corresponding variables in turn until a lower function value is obtained. The normal algorithm is then resumed from the perturbed point.

If a saddle point is suspected, a local search is carried out with a view to moving away from the saddle point. A local search is also performed when a point is found which is thought to be a constrained minimum.

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 **funct2** to calculate the values of the function $F(x)$ and its first derivative $\dfrac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with nag_opt_bounds_quasi_deriv_easy (e04ky) (see the E04 Chapter Introduction).

```
[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_quasi_deriv_easy (e04ky) with the object supplied to nag_opt_bounds_quasi_deriv_easy (e04ky).

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_quasi_deriv_easy (e04ky) 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_quasi_deriv_easy (e04ky) 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: **liw** – INTEGER

Default: **n** $+ 2$

The dimension of the array **iw**.

Constraint: **liw** $\geq$ **n** $+ 2$.

3: **lw** – INTEGER

Default: $\max(10 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2, 11)$

The dimension of the array **w**.

Constraint: **lw** $\geq \max(10 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2, 11)$.

4: **user** – INTEGER array

user is not used by nag_opt_bounds_quasi_deriv_easy (e04ky), 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_quasi_deriv_easy (e04ky).

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

The upper bounds actually used by nag_opt_bounds_quasi_deriv_easy (e04ky).

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

The lowest point found during the calculations. Thus, if **ifail** $= 0$ on exit, **x**(j) is the jth component 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: **iw**(**liw**) – INTEGER array

If **ifail** $= 0$, 3 or 5, the first **n** elements of **iw** contain information about which variables are currently on their bounds and which are free. Specifically, if x_i is:

- fixed on its upper bound, **iw**(i) is -1;
- fixed on its lower bound, **iw**(i) is -2;
- effectively a constant (i.e., $l_j = u_j$), **iw**(i) is -3;
- free, **iw**(i) gives its position in the sequence of free variables.

In addition, **iw**$(\mathbf{n} + 1)$ contains the number of free variables (i.e., n_z). The rest of the array is used as workspace.

7: **w**(**lw**) – REAL (KIND=nag_wp) array

If **ifail** $= 0$, 3 or 5, **w**(i) contains the ith element of the projected gradient vector g_z, for $i = 1, 2, \ldots, \mathbf{n}$. In addition, **w**$(\mathbf{n} + 1)$ contains an estimate of the condition number of the projected Hessian matrix (i.e., k). The rest of the array is used as workspace.

8: **user** – INTEGER array

9: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Note: nag_opt_bounds_quasi_deriv_easy (e04ky) 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** $< \max(11, 10 \times$ **n** $+$ **n** $\times$ (**n** $- 1)/2)$.

ifail $= 2$

There have been $100 \times n$ 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 $= 4$

An overflow has occurred during the computation. This is an unlikely failure, but if it occurs you should restart at the latest point given in **x**.

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_quasi_deriv_easy (e04ky) 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 it may be advisable to try nag_opt_bounds_mod_deriv_easy (e04kz).

7 Accuracy

A successful exit (**ifail** $= 0$) is made from nag_opt_bounds_quasi_deriv_easy (e04ky) when (B1, B2 and B3) or B4 hold, and the local search confirms a minimum, where

$$\text{B1} \equiv \alpha^{(k)} \times \left\|p^{(k)}\right\| < (x_{tol} + \sqrt{\epsilon}) \times \left(1.0 + \left\|x^{(k)}\right\|\right)$$
$$\text{B2} \equiv \left|F^{(k)} - F^{(k-1)}\right| < \left(x_{tol}^2 + \epsilon\right) \times \left(1.0 + \left|F^{(k)}\right|\right)$$
$$\text{B3} \equiv \left\|g_z^{(k)}\right\| < \left(\epsilon^{1/3} + x_{tol}\right) \times \left(1.0 + \left|F^{(k)}\right|\right)$$
$$\text{B4} \equiv \left\|g_z^{(k)}\right\| < 0.01 \times \sqrt{\epsilon}.$$

(Quantities with superscript k are the values at the kth iteration of the quantities mentioned in Section 3, $x_{tol} = 100\sqrt{\epsilon}$, ϵ is the ***machine precision*** and $\|.\|$ denotes the Euclidean norm. The vector g_z is returned in the array **w**.)

If **ifail** $= 0$, then the vector in **x** on exit, x_{sol}, is almost certainly an estimate of the position of the minimum, x_{true}, to the accuracy specified by x_{tol}.

If **ifail** $= 3$ or 5, x_{sol} may still be a good estimate of x_{true}, but the following checks should be made. Let k denote an estimate of the condition number of the projected Hessian matrix at x_{sol}. (The value of k is returned in $\mathbf{w}(\mathbf{n}+1)$). If

(i) the sequence $\left\{F\left(x^{(k)}\right)\right\}$ converges to $F(x_{\text{sol}})$ at a superlinear or a fast linear rate,

(ii) $\|g_z(x_{\text{xol}})\|^2 < 10.0 \times \epsilon$ and

(iii) $k < 1.0/\|g_z(x_{\text{sol}})\|$,

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (ii) is true, then usually $F(x_{\text{sol}})$ is a close approximation to $F(x_{\text{true}})$.

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_quasi_deriv_easy (e04ky) is roughly proportional to n^2. In addition, each iteration makes at least one call of **funct2**. 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 each 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_quasi_deriv_easy (e04ky) 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

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

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

9.1 Program Text

```
function e04ky_example

fprintf('e04ky example results\n\n');

ibound = nag_int(0);
bl = [1;  -2;  -1000000;    1];
bu = [3;   0;   1000000;    3];
x  = [3;  -1;         0;    1];
[bl, bu, x, f, g, iw, w, user, ifail] = ...
  e04ky(ibound, @funct2, bl, bu, x);
fprintf('Minimum point,    x = %7.3f %7.3f %7.3f %7.3f\n',x);
fprintf('At found minimum, f = %7.3f\n',f);
fprintf('                  g = %7.3f %7.3f %7.3f %7.3f\n',g);

function [fc, gc, user] = funct2(n, xc, user)
  gc = zeros(n, 1);

  a = xc(1) + 10*xc(2);
  b = xc(3) -    xc(4);
  c = xc(2) -  2*xc(3);
  d = xc(1) -    xc(4);
  fc = a^2 + 5*b^2 + c^4 + 10*d^4;
  gc(1) =   2*a + 40*d^3;
  gc(2) =  20*a +  4*c^3;
  gc(3) =  10*b -  8*c^3;
  gc(4) = -10*b - 40*d^3;
```

9.2 Program Results

```
e04ky example results

Minimum point,    x =   1.000  -0.085   0.409   1.000
At found minimum, f =   2.434
                  g =   0.295   0.000  -0.000   5.907
```