nag_opt_bounds_no_deriv (e04jbc)

1. Purpose

nag_opt_bounds_no_deriv (e04jbc) is a comprehensive quasi-Newton algorithm for finding:
- an unconstrained minimum of a function of several variables
- a minimum of a function of several variables subject to fixed upper and/or lower bounds on the variables.

No derivatives are required. The function nag_opt_bounds_no_deriv is intended for objective functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2. Specification

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

void nag_opt_bounds_no_deriv(Integer n,
    void (*objfun)(Integer n, double x[], double *objf,
                    double g[], Nag_Comm *comm),
    Nag_BoundType bound, double bl[], double bu[],
    double x[], double *objf, double g[],
    Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

3. Description

This function is applicable to problems of the form:

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

Special provision is made for unconstrained minimization (i.e., 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 \( u_1 = u_2 = \ldots = u_n \). It is possible to specify that a particular \( x_j \) should be held constant. The user must supply a starting point and a function objfun to calculate the value of \( F(x) \) at any point \( x \).

A typical iteration starts at the current point \( x \) where \( n_z \) (say) variables are free from both their bounds. The vector \( g_z \), whose elements are finite-difference approximations to 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 to the matrix of second derivatives with respect to the free variables, are also stored. The equations

\[ LDL^T p_z = -g_z \]

are solved to give a search direction \( p_z \), which is expanded to an \( n \)-vector \( p \) by the insertion of appropriate zero elements. Then \( \alpha \) is found such that \( F(x + \alpha p) \) is approximately a minimum (subject to the fixed bounds) with respect to \( \alpha; \) \( x \) 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 estimated gradient by the step \( \alpha p \). If any variable actually reaches a bound during the search along \( p \), it is fixed and \( n_z \) is reduced for the next iteration. Most iterations calculate \( g_z \) using forward differences, but central differences are used when they seem necessary.

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 is 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. In addition, nag_opt_bounds_no_deriv gives the user the option of specifying that a local search should be performed when a point is found which is thought to be a constrained minimum.

If the user specifies that the problem is unconstrained, nag_opt_bounds_no_deriv sets the \( l_j \) to \(-10^{10}\) and the \( u_j \) to \( 10^{10}\). Thus, provided that the problem has been sensibly scaled, no bounds will be encountered during the minimization process and nag_opt_bounds_no_deriv will act as an unconstrained minimization algorithm. When the problem is unconstrained, the function values used for estimating the first derivatives will always be required in sets of \( n \). nag_opt_bounds_no_deriv enables the user to take advantage (via the parameter \textbf{bound}) of the fact that such sets can often be evaluated in less computer time than \( n \) separate function evaluations would take in general.

4. Parameters

\( n \)

\textbf{n}put: the number \( n \) of independent variables.
\textbf{Constraint}: \( n \geq 1 \).

\textbf{objfun}

\textbf{objfun} must evaluate the function \( F(x) \) at any \( x \). If nag_opt_bounds_no_deriv is called with \textbf{bound} = Nag_NoBounds_One_Call, \textbf{objfun} must also be able to provide the set of \( n \) function values used for estimating first derivatives. (However, if the user does not wish to calculate \( F \) at a particular \( x \), there is the option of setting a parameter to cause nag_opt_bounds_no_deriv to terminate immediately.)

The specification for \textbf{objfun} is:

```c
void objfun(Integer n, double x[], double *objf, double g[], Nag_Comm *comm)
```

\( n \)

\textbf{n}put: the number \( n \) of variables.

\( x[n] \)

\textbf{Input}: the point \( x \) at which the value of \( F \) is required.

\( \textbf{objf} \)

\textbf{Output}: if \textbf{comm}->\textbf{flag} = 0 on entry, then \textbf{objfun} must set \textbf{objf} to the value of the objective function \( F \) at the current point given in \( x \). If it is not possible to evaluate \( F \), then \textbf{objfun} should assign a negative value to \textbf{comm}->\textbf{flag}; nag_opt_bounds_no_deriv will then terminate.

\( g[n] \)

\textbf{Input}: if \textbf{comm}->\textbf{flag} = 3 then \( g \) contains a set of differencing intervals.
\textbf{Output}: if \textbf{comm}->\textbf{flag} = 3 on entry, then \textbf{objfun} must reset \( g[j-1] \) to \( F(x_c + g[j-1] \times e_j) \) for \( j = 1, 2, \ldots, n \), where \( x_c \) is the point given in \( x \) and \( e_j \) is the \( j \)th co-ordinate direction. If it is not possible to evaluate the elements of \( g \) then \textbf{objfun} should assign a negative value to \textbf{comm}->\textbf{flag}; nag_opt_bounds_no_deriv will then terminate.

Thus, since the function values are required at \( n \) points which each differ from \( x_c \) only in one co-ordinate, it may be possible to calculate some terms once but use them in the calculation of more than one function value. (If \textbf{comm}->\textbf{flag} = 0 on entry, \textbf{objfun} must not change the elements of \( g \).)
comm

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

flag – Integer

Input: comm->flag will be set to 0 or 3. The value 0 indicates that a single function value is required. The value 3 (which will only occur if nag_opt_bounds_no_deriv is called with bound set to Nag_NoBounds_One_Call) indicates that a set of n function values is required.

Output: if objfun resets comm->flag to some negative number then nag_opt_bounds_no_deriv will terminate immediately with the error indicator NE_USER_STOP. If fail is supplied to nag_opt_bounds_no_deriv, fail.errnum will be set to the user’s setting of comm->flag.

first – Boolean

Input: will be set to TRUE on the first call to objfun and FALSE for all subsequent calls.

nf – Integer

Input: the number of calculations of the objective function; this value will be equal to the number of calls made to objfun, including the current one, unless the parameter bound = Nag_NoBounds_One_Call.

user – double *

iuser – Integer *

p – Pointer

The type Pointer will be void * with a C compiler that defines void * and char * otherwise. Before calling nag_opt_bounds_no_deriv these pointers may be allocated memory by the user and initialized with various quantities for use by objfun when called from nag_opt_bounds_no_deriv.

Note: objfun should be tested separately before being used in conjunction with nag_opt_bounds_no_deriv. The array x must not be changed by objfun.

bound

Input: indicates whether the problem is unconstrained or bounded. If the problem is unconstrained, the value of bound can be used to indicate that the user wishes objfun to be called with comm->flag set to 3 when a set of n function values is required for making difference estimates of derivatives. If there are bounds on the variables, bound can be used to indicate whether the facility for dealing with bounds of special forms is to be used. bound should be set to one of the following values:

bound = Nag_Bounds

if the variables are bounded and the user will be supplying all the \( l_j \) and \( u_j \) individually.

bound = Nag_NoBounds

if the problem is unconstrained and the user wishes objfun to be called n times with comm->flag set to 0 when a set of function values is required for making difference estimates.

bound = Nag_BoundsZero

if the variables are bounded, but all the bounds are of the form 0 \( \leq \) \( x_j \).

bound = Nag_BoundsEqual

if all the variables are bounded, and \( l_1 = l_2 = \ldots = l_n \) and \( u_1 = u_2 = \ldots = u_n \).

bound = Nag_NoBounds_One_Call

if the problem is unconstrained and the user wishes a single call to be made to objfun with comm->flag = 3 when a set of function values are required for making difference estimates.

Constraint: bound = Nag_Bounds, Nag_NoBounds, Nag_BoundsZero, Nag_BoundsEqual or Nag_NoBounds_One_Call.
nag_opt_bounds_no_deriv

Input: the lower bounds $l_j$.
If bound is set to Nag_Bounds, the user must set $bl[j - 1]$ to $l_j$, for $j = 1, 2, \ldots, n$. (If a lower bound is not required for any $x_j$, the corresponding $bl[j - 1]$ should be set to a large negative number, e.g., $-10^{10}$.)
If bound is set to Nag_BoundsEqual, the user must set $bl[0]$ to $l_1$; nag_opt_bounds_no_deriv will then set the remaining elements of $bl$ equal to $bl[0]$.
If bound is set to Nag_NoBounds, Nag_BoundsZero or Nag_NoBounds_One_Call, $bl$ will be initialized by nag_opt_bounds_no_deriv.
Output: the lower bounds actually used by nag_opt_bounds_no_deriv, e.g., if bound = Nag_BoundsZero, $bl[0] = bl[1] = \ldots = bl[n - 1] = 0.0$.

Input: the upper bounds $u_j$.
If bound is set to Nag_Bounds, the user must set $bu[j - 1]$ to $u_j$, for $j = 1, 2, \ldots, n$. (If an upper bound is not required for any $x_j$, the corresponding $bu[j - 1]$ should be set to a large positive number, e.g., $10^{10}$.)
If bound is set to Nag_BoundsEqual, the user must set $bu[0]$ to $u_1$; nag_opt_bounds_no_deriv will then set the remaining elements of $bu$ equal to $bu[0]$.
If bound is set to Nag_NoBounds, Nag_BoundsZero or Nag_NoBounds_One_Call, $bu$ will be initialized by nag_opt_bounds_no_deriv.
Output: the upper bounds actually used by nag_opt_bounds_no_deriv, e.g., if bound = Nag_BoundsZero, $bu[0] = bu[1] = \ldots = bu[n - 1] = 10^{10}$.

Input: $x[j - 1]$ must be set to a guess at the $j$th component of the position of the minimum, for $j = 1, 2, \ldots, n$.
Output: the final point $x^*$. Thus, if fail.code = NE_NOERROR on exit, $x[j - 1]$ is the $j$th component of the estimated position of the minimum.

Input: if options.init_state is Nag_Init_None (the default) or Nag_Init_H_S, the user need not initialize objf. If options.init_state = Nag_Init_All, objf must be set on entry to the value of $F(x)$ at the initial point supplied by the user in $x$.
Output: the function value at the final point given in $x$.

Input: if options.init_state = Nag_Init_All, $g$ must be set on entry to an approximation to the first derivative vector at the initial $x$. This could be calculated by central differences. If options.init_state is Nag_Init_None or Nag_Init_H_S, $g$ need not be set.
Output: a finite difference approximation to the first derivative vector. Note that the elements of $g$ corresponding to free variables are updated every iteration, but the elements corresponding to fixed variables are only updated when it is necessary to test the Lagrange-multiplier estimates (see Section 3). So, in the printout from nag_opt_bounds_no_deriv (see Section 4.1 and Section 7.3) and on exit from nag_opt_bounds_no_deriv, the elements of $g$ corresponding to fixed variables may be out of date. The elements of $g$ corresponding to free variables should normally be close to zero on exit from nag_opt_bounds_no_deriv.

Input/Output: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_bounds_no_deriv. These structure members offer the means of adjusting some of the parameter values of the algorithm and on output will supply further details of the results. A description of the members of options is given below in Section 7. Some of the results returned in options can be used by nag_opt_bounds_no_deriv to perform a ‘warm start’ if it is re-entered (see the member init_state in Section 7.2).
If any of these optional parameters are required then the structure options should be declared and initialized by a call to nag_opt_init (e04xzc) and supplied as an argument to nag_opt_bounds_no_deriv. However, if the optional parameters are not required the NAG defined null pointer, E04_DEFAULT, can be used in the function call.

Input/Output: structure containing pointers for communication with user-supplied functions;
see the above description of \texttt{objfun} for details. If the user does not need to make use of this communication feature the null pointer \texttt{NAGCOMM\_NULL} may be used in the call to \texttt{nag\_opt\_bounds\_no\_deriv}; \texttt{comm} will then be declared internally for use in calls to user-supplied functions.

\texttt{fail}

The NAG error parameter, see the Essential Introduction to the NAG C Library.

Users are recommended to declare and initialize \texttt{fail} and set \texttt{fail.print} = \texttt{TRUE} for this function.

4.1. Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled by the user with the structure member \texttt{options\_print\_level} (see Section 7.2). The default print level of \texttt{Nag\_Solv\_Iter} provides a single line of output at each iteration and the final result.

This section describes the default printout produced by \texttt{nag\_opt\_bounds\_no\_deriv}.

The following line of output is produced at each iteration. In all cases the values of the quantities printed are those in effect \textit{on completion} of the given iteration.

\begin{itemize}
  \item \texttt{Itn} \quad the iteration count, \(k\).
  \item \texttt{Nfun} \quad the cumulative number of calls made to \texttt{objfun}.
  \item \texttt{Objective} \quad the value of the objective function, \(F(x^{(k)})\).
  \item \texttt{Norm g} \quad the Euclidean norm of the projected gradient vector, \(\|g_z(x^{(k)})\|\).
  \item \texttt{Norm x} \quad the Euclidean norm of \(x^{(k)}\).
  \item \texttt{Norm(x(k-1)-x(k))} \quad the Euclidean norm of \(x^{(k-1)} - x^{(k)}\).
  \item \texttt{Step} \quad the step \(\alpha^{(k)}\) taken along the computed search direction \(p^{(k)}\).
  \item \texttt{Cond H} \quad the ratio of the largest to the smallest element of the diagonal factor \(D\) of the projected Hessian matrix. This quantity is usually a good estimate of the condition number of the projected Hessian matrix. (If no variables are currently free, this value will be zero.)
\end{itemize}

The printout of the final result consists of:

\begin{itemize}
  \item \texttt{x} \quad the final point, \(x^*\).
  \item \texttt{g} \quad the final estimate of the projected gradient vector, \(g_z(x^*)\).
  \item \texttt{Status} \quad the final state of the variable with respect to its bound.
\end{itemize}

5. Comments

A list of possible error exits and warnings from \texttt{nag\_opt\_bounds\_no\_deriv} is given in Section 8. Details of timing, scaling, accuracy and the use of \texttt{nag\_opt\_bounds\_no\_deriv} for unconstrained minimization are given in Section 9.

6. Example 1

This example minimizes the function

\[ F = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \]

subject to the bounds

\begin{align*}
1 & \leq x_1 \leq 3 \\
-2 & \leq x_2 \leq 0 \\
1 & \leq x_4 \leq 3
\end{align*}

starting from the initial guess \((3, -1, 0, 1)^T\).

This example shows the simple use of \texttt{nag\_opt\_bounds\_no\_deriv} where default values are used for all optional parameters. An example showing the use of optional parameters is given in Section 12. There is one example program file, the main program of which calls both examples. The main program and Example 1 are given below.
6.1. Program Text

/* nag_opt_bounds_no_deriv (e04jbc) Example Program
 */

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

#ifdef NAG_PROTO
static void objfun(Integer n, double x[], double *f,
                    double g[], Nag_Comm *comm);
static void ex1(void);
static void ex2(void);
#else
static void objfun();
static void ex1();
static void ex2();
#endif

#define NMAX 4

main()
{
    /* Two examples are called, ex1() which uses the
     * default settings to solve the problem and
     * ex2() which solves the same problem with
     * some optional parameters set by the user.
     */

    Vprintf("e04jbc Example Program Results.\n");
ex1();
ex2();
exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void objfun(Integer n, double x[], double *objf,
                    double g[], Nag_Comm *comm)
#else
static void objfun(n, x, objf, g, comm)
integer n;
double x[];
double *objf;
double g[];
Nag_Comm *comm;
#endif

/* Routine to evaluate objective function. */

double a, b, c, d, x1, x2, x3, x4;
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;
*objf = a*a + 5.0*b*b + c*c + 10.0*d*d;
static void ex1()
{
  double x[NMAX], g[NMAX], bl[NMAX], bu[NMAX];
  double objf;
  Integer n;
  Nag_BoundType bound;
  static NagError fail;

  Vprintf("\ne04jbc example 1: no option setting.\\n");
  fail.print = TRUE;
  n = NMAX;
  x[0] = 3.0;
  x[1] = -1.0;
  x[2] = 0.0;
  x[3] = 1.0;

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

  /* Call optimization routine */
  e04jbc(n, objfun, bound, bl, bu, x, &objf,
         g, E04_DEFAULT, NAGCOMM_NULL, &fail);
  if (fail.code != NE_NOERROR && fail.code != NW_COND_MIN) exit(EXIT_FAILURE);
}

6.2. Program Data

None; but there is an example data file which contains the optional parameter values for Example 2 below.

6.3. Program Results

e04jbc Example Program Results.

e04jbc example 1: no option setting.

Parameters to e04jbc
---------------------
Number of variables........... 4

  optim_tol.................. 1.05e-07  linesearch_tol.......... 5.00e-01
  step_max.................. 1.00e+06  max_iter.................. 200
  init_state.............. Nag_Init_None  local_search............... TRUE
  print_level............. Nag_Soln_Iter  machine precision....... 1.11e-16
  outfile............... stdout

Memory allocation:
  delta.................... Nag
  state................... Nag
  hesl..................... Nag
  hesd..................... Nag

Results from e04jbc:
---------------------
Iteration results:

<table>
<thead>
<tr>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2.1500e+02</td>
<td>1.4e+02</td>
<td>3.3e+00</td>
<td>3.7e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1.6467e+02</td>
<td>3.2e+02</td>
<td>3.2e+00</td>
<td>7.0e-01</td>
<td>1.0e+00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8.7285e+00</td>
<td>4.2e+01</td>
<td>1.4e+00</td>
<td>2.0e+00</td>
<td>6.3e-03</td>
<td>3.6e+00</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>2.9252e+00</td>
<td>7.9e+00</td>
<td>1.4e+00</td>
<td>3.3e-01</td>
<td>1.1e+00</td>
<td>5.0e+00</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>2.4562e+00</td>
<td>1.5e+00</td>
<td>1.5e+00</td>
<td>1.7e-01</td>
<td>1.7e+00</td>
<td>5.3e+00</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>2.4356e+00</td>
<td>4.6e-01</td>
<td>1.5e+00</td>
<td>3.9e-02</td>
<td>1.0e+00</td>
<td>4.2e+00</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>2.4338e+00</td>
<td>4.7e-03</td>
<td>1.5e+00</td>
<td>8.7e-03</td>
<td>1.1e+00</td>
<td>4.4e+00</td>
</tr>
<tr>
<td>7</td>
<td>33</td>
<td>2.4338e+00</td>
<td>9.3e-05</td>
<td>1.5e+00</td>
<td>4.5e-05</td>
<td>1.0e+00</td>
<td>4.4e+00</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>2.4338e+00</td>
<td>6.3e-06</td>
<td>1.5e+00</td>
<td>7.0e-07</td>
<td>1.0e+00</td>
<td>5.2e+00</td>
</tr>
<tr>
<td>9</td>
<td>47</td>
<td>2.4338e+00</td>
<td>5.8e-07</td>
<td>1.5e+00</td>
<td>7.7e-08</td>
<td>1.0e+00</td>
<td>5.2e+00</td>
</tr>
<tr>
<td>10</td>
<td>54</td>
<td>2.4338e+00</td>
<td>2.3e-07</td>
<td>1.5e+00</td>
<td>7.9e-09</td>
<td>1.4e+00</td>
<td>5.2e+00</td>
</tr>
</tbody>
</table>

Local search performed.

Final solution:

<table>
<thead>
<tr>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>74</td>
<td>2.4338e+00</td>
<td>2.3e-07</td>
<td>1.5e+00</td>
<td>0.0e+00</td>
<td>0.0e+00</td>
<td>5.2e+00</td>
</tr>
</tbody>
</table>

Variable x

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower Bound</td>
</tr>
<tr>
<td>Free</td>
</tr>
<tr>
<td>Lower Bound</td>
</tr>
</tbody>
</table>

7. Optional Parameters

A number of optional input and output parameters to nag_opt_bounds_no_deriv are available through the structure argument options, type Nag_E04_Opt. A parameter may be selected by assigning an appropriate value to the relevant structure member; those parameters not selected will be assigned default values. If no use is to be made of any of the optional parameters the user should use the NAG defined null pointer, E04_DEFAULT, in place of options when calling nag_opt_bounds_no_deriv; the default settings will then be used for all parameters.

Before assigning values to options directly the structure must be initialized by a call to the function nag_opt_init(e04xnc). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a text file using the function nag_opt_read(e04xyc) in which case initialization of the options structure will be performed automatically if not already done. Any subsequent direct assignment to the options structure must not be preceded by initialization.

If assignment of functions and memory to pointers in the options structure is required, then this must be done directly in the calling program; they cannot be assigned using using nag_opt_read(e04xyc).

7.1. Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of options which are valid for nag_opt_bounds_no_deriv together with their default values where relevant. The number $\epsilon$ is a generic notation for machine precision (see nag_machine_precision(X02AJC)).

<table>
<thead>
<tr>
<th>Boolean list</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nag_PrintType print_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Soln_Iter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>char outfile[80]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdout</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void (*print_fun)()</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nag_InitType init_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Init_None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integer max_iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>50n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double optim_tol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10\sqrt{\epsilon}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double linesearch_tol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 (0.0 if n = 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double step_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>double f_est</th>
</tr>
</thead>
<tbody>
<tr>
<td>size n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean local_search</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
</tr>
</tbody>
</table>
7.2. Description of Optional Parameters

- **list** – Boolean  
  Default = TRUE  
  Input: if `options.list` = TRUE the parameter settings in the call to nag_opt_bounds_no_deriv will be printed.

- **print_level** – Nag_PrintType  
  Default = Nag_Soln_Init  
  Input: the level of results printout produced by nag_opt_bounds_no_deriv. The following values are available:
  
  - Nag_NoPrint: No output.  
  - Nag_Soln: The final solution.  
  - Nag_Init: One line of output for each iteration.  
  - Nag_Soln_Init: The final solution and one line of output for each iteration.  
  - Nag_Soln_Init_Full: The final solution and detailed printout at each iteration.  

  Details of each level of results printout are described in Section 7.3.

- **outfile** – char[80]  
  Default = stdout  
  Input: the name of the file to which results should be printed. If `options.outfile[0] = '\n'` then the stdout stream is used.

- **print_fun** – pointer to function  
  Default = NULL  
  Input: printing function defined by the user; the prototype of `print_fun` is `void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);`  
  See Section 7.3.1 below for further details.

- **init_state** – Nag_InitType  
  Default = Nag_Init_None  
  Input: `init_state` specifies which of the parameters objf, g, options.hesl, options.hesd and options.state are actually being initialized by the user. Such information will generally reduce the time taken by nag_opt_bounds_no_deriv.

  - **init_state = Nag_Init_None**  
    No values are assumed to have been set in any of objf, g, options.hesl, options.hesd or options.state. (nag_opt_bounds_no_deriv will use the unit matrix as the initial estimate of the Hessian matrix.)

  - **init_state = Nag_Init_All**  
    The parameters objf and g must contain the value of \( F(x) \) and estimates of its first derivatives at the starting point. All \( n \) elements of options.state must have been set to indicate which variables are on their bounds and which are free. The pointer options.delta must give the \( n \) finite-difference intervals. options.hesl and options.hesd must contain the Cholesky factors of a positive-definite approximation to the \( n \times n \) Hessian matrix for the subspace of free variables. (This option is useful for restarting the minimization process if options.max_iter is reached.)

  - **init_state = Nag_Init_H_S**  
    No values are assumed to have been set in objf or g, but options.hesl, options.hesd, options.state and options.delta must have been set as for init_state = Nag_Init_All. (This option is useful for starting off a minimization run using second derivative information from a previous, similar, run.)

  - **init_state** = Nag_Init_None or Nag_Init_All or Nag_Init_H_S.

- **max_iter** – Integer  
  Default = 50n  
  Input: the limit on the number of iterations allowed before termination.

  Constraint: options.max_iter \( \geq 0 \).
optim_tol – double

Input: the accuracy in \(x\) to which the solution is required.
If \(x_{\text{true}}\) is the true value of \(x\) at the minimum, then \(x_{\text{sol}}\), the estimated position prior to a normal exit, is such that
\[
\|x_{\text{sol}} - x_{\text{true}}\| < \text{optim\_tol} \times (1.0 + \|x_{\text{true}}\|),
\]
where \(\|y\| = \sqrt{\sum_{j=1}^{n} y_j^2}\). For example, if the elements of \(x_{\text{sol}}\) are not much larger than 1.0 in modulus and if \text{optim\_tol} is set to \(10^{-5}\), then \(x_{\text{sol}}\) is usually accurate to about 5 decimal places. (For further details see Section 9.3.)

If the problem is scaled roughly as described in Section 9.2 and \(\epsilon\) is the machine precision, then \(\sqrt{\epsilon}\) is probably the smallest reasonable choice for \text{optim\_tol}. (This is because, normally, to machine accuracy, \(F(x + \epsilon e_j) = F(x)\) where \(e_j\) is any column of the identity matrix.)

Constraint: \(\epsilon \leq \text{options\_optim\_tol} < 1.0\).

linesearch_tol – double

Default = 0.5. (If \(n = 1\), default = 0.0)

Input: every iteration of nag\_opt\_bounds\_no\_deriv involves a linear minimization (i.e., minimization of \(F(x + \alpha p\)) with respect to \(\alpha\)). \text{linesearch\_tol} specifies how accurately these linear minimizations are to be performed. The minimum with respect to \(\alpha\) will be located more accurately for small values of \text{linesearch\_tol} (say 0.01) than for large values (say 0.9).

Although accurate linear minimizations will generally reduce the number of iterations performed by nag\_opt\_bounds\_no\_deriv, they will increase the number of function evaluations required for each iteration. On balance, it is usually more efficient to perform a low accuracy linear minimization.

A smaller value such as 0.01 may be worthwhile:

(a) if \(F(x)\) can be evaluated unusually quickly (since it may be worth using extra function evaluations to reduce the number of iterations and associated matrix calculations);

(b) if \(F(x)\) is a penalty or barrier function arising from a constrained minimization problem (since such problems are very difficult to solve).

If \(n = 1\), the default for \text{linesearch\_tol} = 0.0 (if the problem is effectively 1-dimensional then \text{linesearch\_tol} should be set to 0.0 by the user even though \(n > 1\); i.e., if for all except one of the variables the lower and upper bounds are equal).

Constraint: \(0.0 \leq \text{options\_linesearch\_tol} < 1.0\).

step_max – double

Default = 1000000.0

Input: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency a slight overestimate is preferable.) nag\_opt\_bounds\_no\_deriv will ensure that, for each iteration,
\[
\sqrt{\sum_{j=1}^{n} (x_j^{(k)} - x_j^{(k-1)})^2} \leq \text{step\_max},
\]
where \(k\) is the iteration number. Thus, if the problem has more than one solution, nag\_opt\_bounds\_no\_deriv is most likely to find the one nearest the starting point. On difficult problems, a realistic choice can prevent the sequence of \(x^{(k)}\) entering a region where the problem is ill-behaved and can also help to avoid possible overflow in the evaluation of \(F(x)\). However an underestimate of \text{step\_max} can lead to inefficiency.

Constraint: \(\text{options\_step\_max} \geq \text{options\_optim\_tol}\).

f\_est – double

Input: an estimate of the function value at the minimum. This estimate is just used for calculating suitable step lengths for starting linear minimizations off, so the choice is not too critical. However, it is better for \text{f\_est} to be set to an underestimate rather than to an overestimate. If no value is supplied then an initial step length of 1.0 is used, though this may be reduced to ensure that the bounds are not overstepped.
local_search – Boolean
Input: local_search must specify whether or not the user wishes a ‘local search’ to be performed when a point is found which is thought to be a constrained minimum. If local_search = TRUE and either the quasi-Newton direction of search fails to produce a lower function value or the convergence criteria are satisfied, then a local search will be performed. This may move the search away from a saddle point or confirm that the final point is a minimum. If local_search = FALSE there will be no local search when a point is found which is thought to be a minimum.

The amount of work involved in a local search is comparable to twice that required in a normal iteration to minimize $F(x + \alpha p)$ with respect to $\alpha$. For most problems this will be small (relative to the total time required for the minimization).

local_search could be set FALSE if:
- it is known from the physical properties of a problem that a stationary point will be the required minimum;
- a point which is not a minimum could be easily recognized, for example if the value of $F(x)$ at the minimum is known.

delta – double *
Input: suitable step lengths for making difference approximations to the partial derivatives of $F(x)$. If delta is not allocated memory by the user and options.init_state = Nag_Init_None then nag_opt_bounds_no_deriv will allocate memory to delta and assign a suitable set of difference intervals. If delta is allocated memory by the user, i.e., delta is not NULL, and options.init_state = Nag_Init_None then difference intervals are assumed to be supplied by delta. When options.init_state ≠ Nag_Init_None then delta must hold the finite difference intervals; these may be the values output from a previous call to nag_opt_bounds_no_deriv.

If the user wishes to supply difference intervals then the following advice can be given. When the problem is scaled roughly as described in Section 9.2 and $\epsilon$ is the machine precision, values in the range $\sqrt{\epsilon}$ to $\epsilon^{2/3}$ may be suitable.

Otherwise, the user must choose suitable settings, bearing in mind that, when forward differences are used, the approximation is

$$\frac{\partial F}{\partial x_j} = \frac{F(x + \delta[j] \times e_j) - F(x)}{\delta[j]}$$

where $e_j$ is the $j$th co-ordinate direction, for $j = 1, 2, \ldots, n$.

Output: the $n$ finite difference intervals used by nag_opt_bound_deriv. If delta is NULL on entry and options.init_state = Nag_Init_None then memory will have been automatically allocated to delta and suitable values assigned.

Constraints: options.delta[j] ≥ 0.0, $x[j] + \delta[j] \neq x[j]$.

state – Integer *
Input: state need not be set if the default option of options.init_state = Nag_Init_None is used as $n$ values of memory will be automatically allocated by nag_opt_bound_deriv.

If the option init_state = Nag_Init_All or init_state = Nag_Init_H_S has been chosen, state must point to a minimum of $n$ elements of memory. This memory will already be available if the calling program has used the options structure in a previous call to nag_opt_bound_deriv with options.init_state = Nag_Init_None and the same value of $n$.

If a previous call has not been made sufficient memory must be allocated by the user.

When init_state = Nag_Init_All or Nag_Init_H_S then state must specify information about which variables are currently on their bounds and which are free. If $x_j$ is:

(a) fixed on its upper bound, state[j − 1] is −1
(b) fixed on its lower bound, state[j − 1] is −2
(c) effectively a constant (i.e., $l_j = u_j$), state[j − 1] is −3
(d) free, state[j − 1] gives its position in the sequence of free variables.
If options.init_state = Nag_Init_None, state will be initialized by nag_opt_bounds_no_deriv. If options.init_state is set to Nag_Init_All or Nag_Init_H_S, state must be initialized by the user before nag_opt_bounds_no_deriv is called.

Output: state gives information as above about the final point given in x.

hesl – double *
hesd – double *

Default memory = max(n(n - 1)/2, 1)
Default memory = n

Input: hesl and hesd need not be set if the default of options.init_state = Nag_Init_None is used as sufficient memory will be automatically allocated by nag_opt_bounds_no_deriv.

If options.init_state = Nag_Init_All or options.init_state = Nag_Init_H_S has been set then hesl must point to a minimum of max(n(n - 1)/2, 1) elements of memory.

hesd must point to at least n elements of memory if options.init_state = Nag_Init_All or options.init_state = Nag_Init_H_S has been chosen.

The appropriate amount of memory will already be available for hesl and hesd if the calling program has used the options structure in a previous call to nag_opt_bounds_no_deriv with options.init_state = Nag_Init_None and the same value of n. If a previous call has not been made sufficient memory must be allocated by the user.

hesl and hesd are used to store the factors L and D of the current approximation to the matrix of second derivatives with respect to the free variables (see Section 3). (The elements of the matrix are assumed to be ordered according to the permutation specified by the positive elements of state, see above.) hesl holds the lower triangle of L, omitting the unit diagonal, stored by rows. hesd stores the diagonal elements of D. Thus if n_x elements of state are positive, the strict lower triangle of L will be held in the first n_x(n_x - 1)/2 elements of hesl and the diagonal elements of D in the first n_x elements of hesd.

If options.init_state = Nag_Init_None (the default), hesl and hesd will be initialized within nag_opt_bounds_no_deriv to the factors of the unit matrix.

If the user sets options.init_state to Nag_Init_All or Nag_Init_H_S, hesl and hesd must contain on entry the Cholesky factors of a positive-definite approximation to the n_x by n_x matrix of second derivatives for the subspace of free variables as specified by the user’s setting of state.

Output: hesl and hesd hold the factors L and D corresponding to the final point given in x. The elements of hesd are useful for deciding whether to accept the result produced by nag_opt_bounds_no_deriv (see Section 9).

iter – Integer
Output: the number of iterations which have been performed in nag_opt_bounds_no_deriv.

nf – Integer
Output: the number of times the residuals have been evaluated.

7.3. Description of Printed Output

The level of printed output can be controlled with the structure options.list and options.print_level (see Section 7.2). If list = TRUE then the parameter values to nag_opt_bounds_no_deriv are listed, whereas the printout of results is governed by the value of print_level. The default of print_level = Nag_Soln_Iter provides a single line of output at each iteration and the final result. This section describes all of the possible levels of printout available from nag_opt_bounds_no_deriv.

When print_level = Nag_Iter or Nag_Soln_Iter a single line of output is produced on completion of each iteration, this gives the following values:

In
the iteration count, k.

Nf
the cumulative number of objective function evaluations.

Objective
the value of the objective function, F(x(k)).

Norm g
de the Euclidean norm of the projected gradient vector, \|g_z(x(k))\|.

Norm x
de the Euclidean norm of x(k).

Norm(x(k-1)-x(k))
de the Euclidean norm of x(k-1) – x(k).

Step
de the step ρ(k) taken along the computed search direction p(k).

Cond H
de the ratio of the largest to the smallest element of the diagonal factor D of the projected Hessian matrix. This quantity is usually a good
estimate of the condition number of the projected Hessian matrix.
(If no variables are currently free, this value will be zero.)

When \texttt{options.print\_level} = \texttt{Nag\_Soln\_Iter\_Full} more detailed results are given at each iteration. Additional values output are
\begin{align*}
x & \quad \text{the current point } x^{(k)}. \\
g & \quad \text{the current estimate of the projected gradient vector, } g_z(x^{(k)}). \\
\text{Status} & \quad \text{the current state of the variable with respect to its bound(s).}
\end{align*}

If \texttt{options.print\_level} = \texttt{Nag\_Soln} or \texttt{Nag\_Soln\_Iter} or \texttt{Nag\_Soln\_Iter\_Full} the final result is printed out. This consists of:
\begin{align*}
x & \quad \text{the final point, } x^*. \\
g & \quad \text{the final estimate of the projected gradient vector, } g_z(x^*). \\
\text{Status} & \quad \text{the final state of the variable with respect to its bound(s).}
\end{align*}

If \texttt{options.print\_level} = \texttt{Nag\_NoPrint} then printout will be suppressed; the user can print the final solution when \texttt{nag\_opt\_bounds\_no\_deriv} returns to the calling program.

### 7.3.1. Output of Results via a User-defined Printing Function

Users may also specify their own print function for output of iteration results and the final solution by use of the \texttt{options.print\_fun} function pointer, which has prototype

void (*print\_fun)(const Nag\_Search\_State *st, Nag\_Comm *comm);

The rest of this section can be skipped if the default printing facilities provide the required functionality.

When a user defined function is assigned to \texttt{options.print\_fun} this will be called in preference to the internal print function of \texttt{nag\_opt\_bounds\_no\_deriv}. Calls to the user defined function are again controlled by means of the \texttt{options.print\_level} member. Information is provided through \texttt{st} and \texttt{comm}, the two structure arguments to \texttt{print\_fun}.

The results contained in the members of \texttt{st} are those on completion of the last iteration or those after a local search. (An iteration may be followed by a local search (see \texttt{options.local\_search}, Section 7.2) in which case \texttt{print\_fun} is called with the results of the last iteration (\texttt{st.local\_search} = \texttt{FALSE}) and then again when the local search has been completed (\texttt{st.local\_search} = \texttt{TRUE}).

If \texttt{comm->it\_prt} = \texttt{TRUE} then the results on completion of an iteration of \texttt{nag\_opt\_bounds\_no\_deriv} are contained in the members of \texttt{st}. If \texttt{comm->sol\_prt} = \texttt{TRUE} then the final results from \texttt{nag\_opt\_bounds\_no\_deriv}, including details of the final iteration, are contained in the members of \texttt{st}. In both cases, the same members of \texttt{st} are set, as follows:

- \texttt{iter} – Integer
  - the current iteration count, \( k \), if \texttt{comm->it\_prt} = \texttt{TRUE}; the final iteration count, \( k \), if \texttt{comm->sol\_prt} = \texttt{TRUE}.
- \texttt{n} – Integer
  - the number of variables.
- \texttt{x} – double *
  - the co-ordinates of the point \( x^{(k)} \).
- \texttt{f} – double
  - the value of the current objective function.
- \texttt{g} – double *
  - the estimated value of \( \partial F/\partial x_j \) at \( x^{(k)} \), \( j = 1, 2, \ldots, n \).
- \texttt{gpj\_norm} – double
  - the Euclidean norm of the current estimate of the projected gradient \( g_z \).
- \texttt{step} – double
  - the step \( \alpha^{(k)} \) taken along the search direction \( p^{(k)} \).
- \texttt{cond} – double
  - the estimate of the condition number of the Hessian matrix.
xk_norm – double
  the Euclidean norm of \( x^{(k-1)} - x^{(k)} \).

state – Integer *
  the status of variables \( x_j, j = 1, 2, \ldots, n \), with respect to their bounds. See Section 7.2 for a
description of the possible status values.

local_search – Boolean
  TRUE if a local search has been performed.

nf – Integer
  the cumulative number of objective function evaluations.

The relevant members of the structure comm are:

itprt – Boolean
  will be TRUE when the print function is called with the results of the current iteration.

solprt – Boolean
  will be TRUE when the print function is called with the final result.

user – double *

iuser – Integer *

p – Pointer
  pointers for communication of user information. If used they must be allocated memory by
  the user either before entry to nag_opt_bounds_no_deriv or during a call to objfun or print_fun.
The type Pointer will be void * with a C compiler that defines void * and char * otherwise.

8. Error Indications and Warnings

NE_USER_STOP
  User requested termination, user flag value = ⟨value⟩.
  This exit occurs if the user sets comm->flag to a negative value in objfun. If fail is supplied
  the value of fail.errnum will be the same as the user's setting of comm->flag.

NE_INT_ARG_LT
  On entry, n must not be less than 1: n = ⟨value⟩.

NE_BOUND
  The lower bound for variable ⟨value⟩ (array element bl[⟨value⟩]) is greater than the upper
  bound.

NE_OPT_NOT_INIT
  Options structure not initialized.

NE_BAD_PARAM
  On entry, parameter bound had an illegal value.
  On entry, parameter options.print_level had an illegal value.
  On entry, parameter options.init_state had an illegal value.

NE_REAL_ARG_LT
  On entry, options.step_max = ⟨value⟩ while options.optim_tol = ⟨value⟩. These parameters
  must satisfy step_max ≥ optim_tol.

NE_INVALID_INT_RANGE_1
  Value ⟨value⟩ given to options.max_iter is not valid.
  Correct range is max_iter ≥ 0.

NE_INVALID_REAL_RANGE_EF
  Value ⟨value⟩ given to options.optim_tol is not valid.
  Correct range is \( \epsilon \leq \text{optim}_\text{tol} < 1.0 \).

NE_INVALID_REAL_RANGE_FF
  Value ⟨value⟩ given to options.linesearch_tol is not valid.
  Correct range is 0.0 ≤ linesearch_tol < 1.0.
NE_NO_MEM
Option init.state = ⟨string⟩ but at least one of the pointers ⟨string⟩ in the option structure has not been allocated memory.

NE_FD_INTERVAL
Finite difference interval for variable ⟨value⟩ (array element options.delta[⟨value⟩]) is negative or so small that x + interval = x.

NE_HESD
The initial values of the supplied options.hesd has some value(s) which is negative or too small or the ratio of the largest element of hesd to the smallest is too large.

NE_ALLOC_FAIL
Memory allocation failed.

When one of the above exits occurs, no values will have been assigned by nag_opt_bounds_no_deriv to objf or to the elements of g, options.hesl, or options.hesd.

NW_TOO_MANY_ITER
The maximum number of iterations, ⟨value⟩, have been performed.

If steady reductions in F(x), were monitored up to the point where this exit occurred, then the exit probably occurred simply because options.max_iter was set too small, so the calculations should be restarted from the final point held in x. This exit may also indicate that F(x) has no minimum.

NW_COND_MIN
The conditions for a minimum have not all been satisfied, but a lower point could not be found.

Provided, on exit, that the estimated first derivatives of F(x) with respect to the free variables are sufficiently small, and that the estimated condition number of the second derivative matrix is not too large, this error exit may simply mean that, although it has not been possible to satisfy the specified requirements, the algorithm has in fact found the minimum as far as the accuracy of the machine permits. This could be because options.optim_tol has been set so small that rounding error in objfun makes attainment of the convergence conditions impossible.

If the estimated condition number of the approximate Hessian matrix at the final point is large, it could be that the final point is a minimum but that the smallest eigenvalue of the second derivative matrix is so close to zero that it is not possible to recognize the point as a minimum.

NE_CHOLESKY_OVERFLOW
An overflow would have occurred during the updating of the Cholesky factors if the calculations had been allowed to continue. Restart from the current point with options.init.state = Nag_Init_None.

NW_LOCAL_SEARCH
The local search has failed to find a feasible point which gives a significant change of function value.

If the problem is a genuinely unconstrained one, this type of exit indicates that the problem is extremely ill conditioned or that the function has no minimum. If the problem has bounds which may be close to the minimum, it may just indicate that steps in the subspace of free variables happened to meet a bound before they changed the function value.

NE_NOT_APPEND_FILE
Cannot open file ⟨string⟩ for appending.

NE_WRITE_ERROR
Error occurred when writing to file ⟨string⟩.
NE_NOT_CLOSE_FILE

Cannot close file \textit{\textless}string\textit{\textgreater}.

An exit of \texttt{fail.code = NW\_TOO\_MANY\_ITER}, \texttt{NW\_COND\_MIN} or \texttt{NW\_LOCAL\_SEARCH} may also be caused by mistakes in \texttt{objfun}, by the formulation of the problem or by an awkward function. If there are no such mistakes, 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.

9. Further Comments

9.1. Timing

The number of iterations required depends on the number of variables, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed in an iteration of \texttt{nag\_opt\_bounds\_no\_deriv} is roughly proportional to $n_z^2$. In addition, each iteration makes at least $n_z + 1$ function evaluations. So, unless $F(x)$ can be evaluated very quickly, the run time will be dominated by the time spent in \texttt{objfun}.

9.2. Scaling

Ideally, the problem should be scaled so that, at the solution, $F(x)$ and the corresponding values of the $x_j$ are each in the range $(-1, +1)$, and so that at points one unit away from the solution, $F(x)$ differs from its value at the solution by approximately one unit. This will usually imply that the Hessian matrix at the solution is well conditioned. It is unlikely that the user 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 \texttt{nag\_opt\_bounds\_no\_deriv} will take less computer time.

9.3. Accuracy

A successful exit (\texttt{fail.code = NE\_NOERROR}) is made from \texttt{nag\_opt\_bounds\_no\_deriv} when (B1, B2 and B3) or B4 hold, and the local search (if used) confirms a minimum, where

\begin{align*}
B1 & \equiv \alpha(k) \times \|p(k)\| < (\text{optim\_tol} + \sqrt{\epsilon}) \times (1.0 + \|x(k)\|) \\
B2 & \equiv \|F(k) - F(k-1)\| < (\text{optim\_tol}^2 + \epsilon) \times (1.0 + \|F(k)\|) \\
B3 & \equiv \|g_z(k)\| < (\epsilon^{1/3} + \text{optim\_tol}) \times (1.0 + \|F(k)\|) \\
B4 & \equiv \|g_z(k)\| < 0.01 \times \sqrt{\epsilon}.
\end{align*}

(Quantities with superscript $k$ are the values at the $k$th iteration of the quantities mentioned in Section 3; $\epsilon$ is the \textit{machine precision}, $\|.\|$ denotes the Euclidean norm and \texttt{optim\_tol} is described in Section 7.)

If \texttt{fail.code = NE\_NOERROR}, 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 \texttt{optim\_tol}.

If \texttt{fail.code = NW\_COND\_MIN} or \texttt{NW\_LOCAL\_SEARCH}, $x_{sol}$ may still be a good estimate of $x_{true}$, but the following checks should be made. Let the largest of the first $n_z$ elements of \texttt{options.hesd} be $\texttt{hesd}[b]$, let the smallest be $\texttt{hesd}[s]$, and define $k = \texttt{hesd}[b]/\texttt{hesd}[s]$. The scalar $k$ is usually a good estimate of the condition number of the projected Hessian matrix at $x_{sol}$. If

(a) the sequence $\{F(x(k))\}$ converges to $F(x_{sol})$ at a superlinear or a fast linear rate,
(b) $\|g_z(x_{sol})\|^2 < 10.0 \times \epsilon$, and
(c) $k < 1.0/\|g_z(x_{sol})\|$, then it is almost certain that $x_{sol}$ is a close approximation to the position of a minimum. When (b) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$. The quantities needed for these checks are all available in the results printout from \texttt{nag\_opt\_bounds\_no\_deriv}; in particular the final value of \texttt{Cond} $H$ gives $k$.

Further suggestions about confirmation of a computed solution are given in the Chapter Introduction.

3.e04jbc.16 [NP3275/5/pdf]
9.4. Unconstrained Minimization

If a problem is genuinely unconstrained and has been scaled sensibly, the following points apply:

(a) \( n_z \) will always be \( n \),
(b) if \texttt{options.init state} is set to \texttt{Nag_Init_All} or \texttt{Nag_Init_HS} on entry, \texttt{options.state}[j - 1] has simply to be set to \( j \), for \( j = 1, 2, \ldots, n \),
(c) \texttt{options.hesl} and \texttt{options.hesd} will be factors of the full approximate second derivative matrix with elements stored in the natural order,
(d) the elements of \( g \) should all be close to zero at the final point,
(e) the \texttt{Status} values given in the printout from \texttt{nag_opt_bounds_no_deriv} and in \texttt{options.state} on exit are unlikely to be of interest (unless they are negative, which would indicate that the modulus of one of the \( x_j \) has reached 10^{10} for some reason),
(f) \texttt{Norm g} simply gives the norm of the estimated first derivative vector.

10. References

Gill P E and Murray W (1976) Minimization subject to bounds on the variables \textit{National Physical Laboratory Report NAC 72}.

11. See Also

\texttt{nag_opt_bounds_deriv (e04kbc)}
\texttt{nag_opt_init (e04xxc)}
\texttt{nag_opt_read (e04xyc)}
\texttt{nag_opt_free (e04xzc)}

12. Example 2

Example 2 solves the same problem as Example 1 but shows the use of certain optional parameters. This example shows option values being assigned directly within the program text and by reading values from a data file. The \texttt{options} structure is declared and initialized by \texttt{nag_opt_init (e04xxc)}, an array holding the required finite difference intervals is then assigned to \texttt{options.delta} and three further options are read from the data file by use of \texttt{nag_opt_read (e04xyc)}. The memory freeing function \texttt{nag_opt_free (e04xzc)} is used to free the memory assigned to the pointers in the option structure. Users should not use the standard C function \texttt{free()} for this purpose.

12.1. Program Text

```c
static void ex2()
{
    double x[NMAX], g[NMAX], bl[NMAX], bu[NMAX];
    Boolean print;
    double objf;
    Integer n;
    Nag_BoundType bound;
    Nag_E04_Opt options;
    static NagError fail, fail2;
    Vprintf("\n\n\e04jbc example 2: using option setting.\n\n");
    fail.print = TRUE;
    fail2.print = TRUE;
```
n = NMAX;
x[0] = 3.0;
x[1] = -1.0;
x[2] = 0.0;
x[3] = 1.0;

/* Initialize options structure */
e04xxc(&options);

options.optim_tol = 100*sqrt(X02AJC); /* Specify accuracy of solution */

/* Read remaining option values from file */
print = TRUE;
e04xyc("e04jbc", "stdin", &options, print, "stdout", &fail);

if (fail.code == NE_NOERROR)
{
    /* Set bounds on variables */
    bound = Nag_Bounds;
    bl[0] = 1.0;
    bu[0] = 3.0;
    bl[1] = -2.0;
    bu[1] = 0.0;
    /* x[2] is not bounded, so we set bl[2] to a large negative
    * number and bu[2] to a large positive number
    */
    bl[2] = -1.0e10;
    bu[2] = 1.0e10;
    bl[3] = 1.0;
    bu[3] = 3.0;

    /* Call optimization routine */
e04jbc(n, objfun, bound, bl, bu, x, &objf,
           g, &options, NAGCOMM_NULL, &fail);

    /* Free Nag allocated memory */
e04xzc(&options, "all", &fail2);
}
if ((fail.code != NE_NOERROR && fail.code != NW_COND_MIN)
    || fail2.code != NE_NOERROR) exit(EXIT_FAILURE);

12.2. Program Data

e04jbc Example Program Data

Following options for e04jbc are read by e04xyc in example 2.

begin e04jbc
print_level = Nag_Soln_Iter_Full /* Print full iterations and solution. */
max_iter = 40 /* Perform maximum of 40 iterations */
step_max = 4.0 /* estimate minimum within 4 units of start */
end

12.3. Program Results

e04jbc example 2: using option setting.

Optional parameter setting for e04jbc.
---------------------------------------

Option file: stdin
print_level set to Nag_Soln_Iter_Full
max_iter set to 40
step_max set to 4.00e+00

Parameters to e04jbc
Number of variables............ 4
optim_tol.................... 1.05e-06
step_max.................. 4.00e+00
init_state..............Nag_Init_None
print_level..............Nag_Soln_Iter_Full
linesearch_tol.......... 5.00e-01
max_iter.................. 40
local_search........... TRUE
machine precision....... 1.11e-16

Memory allocation:
delta...................... Nag
state.................... Nag
hesl..................... Nag
hesd.................... Nag

Results from e04jbc:

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.1500e+02</td>
<td>1.4e+02</td>
<td>3.3e+00</td>
<td>3.7e+00</td>
<td></td>
<td>3.7e+00</td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.0000e+00</td>
<td>3.0600e+02</td>
<td>Upper Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-1.0000e+00</td>
<td>-1.4400e+02</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.0000e+00</td>
<td>-2.0000e+00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>-3.1000e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1.6467e+02</td>
<td>3.2e+02</td>
<td>3.2e+00</td>
<td>7.0e-01</td>
<td>1.0e+00</td>
<td>2.1e+02</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.0000e+00</td>
<td>3.1997e+02</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-3.0174e-01</td>
<td>-5.5533e-01</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.5448e-02</td>
<td>-9.2150e-00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>-3.1035e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>8.7285e+00</td>
<td>4.2e+01</td>
<td>1.4e+00</td>
<td>2.0e+00</td>
<td>6.3e-03</td>
<td>3.6e+00</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>3.1997e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-3.0166e-01</td>
<td>5.6219e+00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.6453e-01</td>
<td>-5.6009e-00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>-3.1035e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>13</td>
<td>2.9252e+00</td>
<td>7.9e+00</td>
<td>1.4e+00</td>
<td>3.3e-01</td>
<td>1.1e+00</td>
<td>5.0e+00</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>3.1997e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-6.7556e-02</td>
<td>5.6219e+00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.6655e-01</td>
<td>-5.6009e-00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>-3.1035e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>17</td>
<td>2.4582e+00</td>
<td>1.5e+00</td>
<td>1.5e+00</td>
<td>1.7e-01</td>
<td>1.7e+00</td>
<td>5.3e+00</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>3.1997e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8.2099e-02</td>
<td>2.7295e-02</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4.3958e-01</td>
<td>1.5017e+00</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>-3.1035e+02</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration results:</th>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm(x(k-1)-x(k))</th>
<th>Step</th>
<th>Cond H</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>22</td>
<td>2.4356e+00</td>
<td>4.6e-01</td>
<td>1.5e+00</td>
<td>3.9e-02</td>
<td>1.0e+00</td>
<td>4.2e+00</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>2.9810e-01</td>
<td>Lower Bound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Itn</td>
<td>Nfun</td>
<td>Objective</td>
<td>Norm g</td>
<td>Norm x</td>
<td>Norm((x(k-1)-x(k)))</td>
<td>Step</td>
<td>Cond</td>
<td>H</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-----------</td>
<td>--------</td>
<td>--------</td>
<td>------------------------</td>
<td>------</td>
<td>------</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>2.4338e+00</td>
<td>4.7e-03</td>
<td>1.5e+00</td>
<td>8.7e-03</td>
<td>1.1e+00</td>
<td>4.4e+00</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>33</td>
<td>2.4338e+00</td>
<td>9.3e-05</td>
<td>1.5e+00</td>
<td>4.5e-05</td>
<td>1.0e+00</td>
<td>4.4e+00</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>2.4338e+00</td>
<td>6.3e-06</td>
<td>1.5e+00</td>
<td>7.0e-07</td>
<td>1.0e+00</td>
<td>4.4e+00</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>47</td>
<td>2.4338e+00</td>
<td>1.4e-07</td>
<td>1.5e+00</td>
<td>7.5e-08</td>
<td>1.0e+00</td>
<td>4.4e+00</td>
<td></td>
</tr>
</tbody>
</table>

Variable | x | g | Status |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>2.9535e-01</td>
<td>Lower Bound</td>
</tr>
<tr>
<td>2</td>
<td>-8.5232e-02</td>
<td>8.5739e-05</td>
<td>Free</td>
</tr>
<tr>
<td>3</td>
<td>4.0930e-01</td>
<td>-3.6669e-05</td>
<td>Free</td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>5.9070e+00</td>
<td>Lower Bound</td>
</tr>
</tbody>
</table>

Local search performed.

Final solution:

<table>
<thead>
<tr>
<th>Itn</th>
<th>Nfun</th>
<th>Objective</th>
<th>Norm g</th>
<th>Norm x</th>
<th>Norm((x(k-1)-x(k)))</th>
<th>Step</th>
<th>Cond</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>64</td>
<td>2.4338e+00</td>
<td>1.4e-07</td>
<td>1.5e+00</td>
<td>0.0e+00</td>
<td>0.0e+00</td>
<td>4.4e+00</td>
<td></td>
</tr>
</tbody>
</table>

Variable | x | g | Status |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000e+00</td>
<td>2.9535e-01</td>
<td>Lower Bound</td>
</tr>
<tr>
<td>2</td>
<td>-8.5232e-02</td>
<td>1.2801e-07</td>
<td>Free</td>
</tr>
<tr>
<td>3</td>
<td>4.0930e-01</td>
<td>4.9100e-08</td>
<td>Free</td>
</tr>
<tr>
<td>4</td>
<td>1.0000e+00</td>
<td>5.9070e+00</td>
<td>Lower Bound</td>
</tr>
</tbody>
</table>