nag_opt_lsq_deriv (e04gbc)

1. Purpose

nag_opt_lsq_deriv (e04gbc) is a comprehensive algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables ($m \geq n$). First derivatives are required.

The function nag_opt_lsq_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

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

void nag_opt_lsq_deriv(Integer m, Integer n,
                       void (*lsqfun)(Integer m, Integer n, double x[], double fvec[],
                                      double fjac[], Integer tdj, Nag_Comm *comm),
                       double x[], double *fsumsq, double fvec[], double fjac[],
                       Integer tdj, Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

3. Description

This function is applicable to problems of the form:

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^T$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.) The user must supply a C function to calculate the values of the $f_i(x)$ and their first derivatives $\partial f_i / \partial x_j$ at any point x.

From a starting point $x^{(1)}$, supplied by the user, nag_opt_lsq_deriv generates a sequence of points $x^{(2)}, x^{(3)}, \ldots$, which is intended to converge to a local minimum of $F(x)$. The sequence of points is given by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$$

where the vector $p^{(k)}$ is a direction of search, and $\alpha^{(k)}$ is chosen such that $F(x^{(k)} + \alpha^{(k)} p^{(k)})$ is approximately a minimum with respect to $\alpha^{(k)}$.

The vector $p^{(k)}$ used depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced, then $p^{(k)}$ is the Gauss-Newton direction; otherwise the second derivatives of the $f_i(x)$ are taken into account using a quasi-Newton updating scheme.

The method is designed to ensure that steady progress is made whatever the starting point, and to have the rapid ultimate convergence of Newton's method.

4. Parameters

m

Input: the number m of residuals, $f_i(x)$

n

Input: the number n of variables, x_j.
Constraint: $\mathbf{1} \leq \mathbf{n} \leq \mathbf{m}$.

lsqfun

The function **lsqfun**, supplied by the user, must calculate the vector of values $f_i(x)$ and their first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. (However, if the user does not wish to calculate the residuals at a particular x, there is the option of setting a parameter to cause nag_opt_lsq_deriv to terminate immediately.)

The specification of **lsqfun** is:

```
void lsqfun(Integer m, Integer n, double x[], double fvec[],
            double fjac[], Integer tdj, Nag_Comm *comm)
```

m
n
Input: the numbers m and n of residuals and variables, respectively.

x[n]
Input: the point x at which the values of the f_i and the $\partial f_i/\partial x_j$ are required.

fvec[m]
Output: unless **comm->flag** $= 1$ on entry, or **comm->flag** is reset to a negative number, then **fvec**$[i-1]$ must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

fjac[m*tdj]
Output: unless **comm->flag** $= 0$ on entry, or **comm->flag** is reset to a negative number, then **fjac**$[(i-1)$*$\mathbf{tdj} + j-1]$ must contain the value of the first derivative $\partial f_i/\partial x_j$ at the point x, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

tdj
Input: the second dimension of the array **fjac** as declared in the function from which nag_opt_lsq_deriv is called.

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

flag – Integer
Input: **comm->flag** contains 0, 1 or 2. The value 0 indicates that only the residuals need to be evaluated, the value 1 indicates that only the Jacobian matrix needs to be evaluated, and the value 2 indicates that both the residuals and the Jacobian matrix must be calculated. (If the default value of the optional parameter **options.minlin** is used, **Nag_Lin_Deriv**, then **lsqfun** will always be called with **comm->flag** set to 2.)
Output: if **lsqfun** resets **comm->flag** to some negative number then nag_opt_lsq_deriv will terminate immediately with the error indicator **NE_USER_STOP**. If **fail** is supplied to nag_opt_lsq_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 **lsqfun** and **FALSE** for all subsequent calls.

nf – Integer
Input: the number of calls made to **lsqfun** including the current one.

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_lsq_deriv these pointers may be allocated memory by the user and initialised with various quantities for use by **lsqfun** when called from nag_opt_lsq_deriv.

Note: **lsqfun** should be tested separately before being used in conjunction with nag_opt_lsq_deriv. Function nag_opt_lsq_check_deriv (e04yac) may be used to check the derivatives. The array **x** must **not** be changed within **lsqfun**.

x[n]
Input: $\mathbf{x}[j-1]$ must be set to a guess at the jth component of the position of the minimum, for $j = 1, 2, \ldots, n$.

Output: the final point x^*. On successful exit, $\mathbf{x}[j-1]$ is the jth component of the estimated position of the minimum.

fsumsq
Output: the value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in **x**.

fvec[m]
Output: **fvec**$[i-1]$ is the value of the residual $f_i(x)$ at the final point given in **x**, for $i = 1, 2, \ldots, m$.

fjac[m][tdj]
Output: **fjac**$[i-1][j-1]$ contains the value of the first derivative $\partial f_i/\partial x_j$ at the final point given in **x**, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

tdj
Input: the second dimension of the array **fjac** as declared in the function from which nag_opt_lsq_deriv is called.
Constraint: **tdj** $\geq$ **n**.

options
Input/Output: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_lsq_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.

If any of these optional parameters are required then the structure **options** should be declared and initialised by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_opt_lsq_deriv. However, if the optional parameters are not required the NAG defined null pointer, `E04_DEFAULT`, can be used in the function call.

comm
Input/Output: structure containing pointers for communication to user-supplied functions; see the above description of **lsqfun** for details. If the user does not need to make use of this communication feature the null pointer `NAGCOMM_NULL` may be used in the call to nag_opt_lsq_deriv; **comm** will then be declared internally for use in calls to user-supplied functions.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.
Users are recommended to declare and initialise **fail** and set **fail.print** = **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 option **print_level** (see Section 7.2.).The default print level of **Nag_Soln_Iter** provides a single line of output at each iteration and the final result.
The line of results printed at each iteration gives:

`Itn`	the current iteration number k.
`Nfun`	the cumulative number of calls to **lsqfun**.
`Objective`	the current value of the objective function, $F(x^{(k)})$.
`Norm g`	the Euclidean norm of the gradient of $F(x^{(k)})$.
`Norm x`	the Euclidean norm of $x^{(k)}$.
`Norm(x(k-1)-x(k))`	the Euclidean norm of $x^{(k-1)} - x^{(k)}$.
`Step`	the step $\alpha^{(k)}$ taken along the computed search direction $p^{(k)}$.

The printout of the final result consists of:

`x`	the final point x^*.
`g`	the gradient of F at the final point.
`Residuals`	the values of the residuals f_i at the final point.
`Sum of squares`	the value of $F(x^*)$, the sum of squares of the residuals at the final point.

5. Comments

A list of possible error exits and warnings from nag_opt_lsq_deriv is given in Section 8.

5.1. Preliminary comments on accuracy

If the problem is reasonably well scaled and 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 the components of x and between $t-1$ (if $F(x)$ is of order 1 at the minimum) and $2t-2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

Further details about accuracy are given in Section 9.

6. Example 1

To find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The program uses (0.5, 1.0, 1.5) as the initial guess at the position of the minimum.

This example shows the simple use of nag_opt_lsq_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 however only 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_lsq_deriv (e04gbc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#ifdef NAG_PROTO
static void lsqfun1(Integer m, Integer n, double x[], double fvec[],
                    double fjac[], Integer tdj, Nag_Comm *comm);
static void lsqfun2(Integer m, Integer n, double x[], double fvec[],
```

```
                    double fjac[], Integer tdj, Nag_Comm *comm);
static void ex1(void);
static void ex2(void);
#else
static void lsqfun1();
static void lsqfun2();
static void ex1();
static void ex2();
#endif

#define MMAX 15
#define NMAX 3
#define TMAX 3

/* Define a user structure template to store data in lsqfun. */
struct user
{
  double y[MMAX];
  double t[MMAX][TMAX];
};

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("e04gbc Example Program Results.\n");
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}

static void ex1()
{
  double fjac[MMAX][NMAX], fvec[MMAX], x[NMAX];
  Integer m, n, tdj;
  double fsumsq;
  static NagError fail;

  Vprintf("\ne04gbc example 1: no option setting.\n");
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  n = 3;
  m = 15;
  tdj = NMAX;

  /* Set up the starting point */
  x[0] = 0.5;
  x[1] = 1.0;
  x[2] = 1.5;

  /* Call the optimization routine */
  fail.print = TRUE;
  e04gbc(m, n, lsqfun1, x, &fsumsq, fvec, (double *)fjac, tdj,
         E04_DEFAULT, NAGCOMM_NULL, &fail);
  if (fail.code != NE_NOERROR && fail.code != NW_COND_MIN) exit(EXIT_FAILURE);
} /* ex1 */

#ifdef NAG_PROTO
static void lsqfun1(Integer m, Integer n, double x[], double fvec[],
                    double fjac[], Integer tdj, Nag_Comm *comm)
#else
     static void lsqfun1(m, n, x, fvec, fjac, tdj, comm)
     Integer m, n;
     double x[], fvec[], fjac[];
     Integer tdj;
     Nag_Comm *comm;
```

```
#endif
{
  /* Function to evaluate the residuals and their 1st derivatives
   * in example 1.
   *
   * This function is also suitable for use when Nag_Lin_NoDeriv is
   * specified for linear minimization instead of the default of
   * Nag_Lin_Deriv, since it can deal with comm->flag = 0 or 1 as
   * well comm->flag = 2.
   *
   * In this example a static variable is used to hold the
   * initial observations. The data is read into the structure
   * gs on the first call to lsqfun1(), it could alternatively
   * be read in from within main().
   */
#define FJAC(I,J) fjac[(I)*tdj + (J)]

  Integer i, j, nt;
  double denom, dummy;

  static struct user gs;

  if (comm->first)
    {
      /* First call to lsqfun(), read data into structure.
       * Observations t (j = 0, 1, 2) are held in gs.t[i][j]
       * (i = 0, 1, 2, . . . , 14)
       */
      nt = 3;
      for (i = 0; i < m; ++i)
        {
          Vscanf("%lf", &gs.y[i]);
          for (j = 0; j < nt; ++j) Vscanf("%lf", &gs.t[i][j]);
        }
    }

  for (i = 0; i < m; ++i)
    {
      denom = x[1]*gs.t[i][1] + x[2]*gs.t[i][2];
      if (comm->flag != 1)
        fvec[i] = x[0] + gs.t[i][0]/denom - gs.y[i];
      if (comm->flag != 0)
        {
          FJAC(i,0) = 1.0;
          dummy = -1.0 / (denom * denom);
          FJAC(i,1) = gs.t[i][0]*gs.t[i][1]*dummy;
          FJAC(i,2) = gs.t[i][0]*gs.t[i][2]*dummy;
        }
    }
}                               /* lsqfun1 */
```

6.2. Program Data

```
e04gbc Example Program Data

Example data for ex1: no option setting
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

6.3. Program Results

```
e04gbc Example Program Results.

e04gbc example 1: no option setting.

Parameters to e04gbc
--------------------

Number of residuals.......... 15    Number of variables..........  3

minlin.............Nag_Lin_Deriv    machine precision....... 1.11e-16
optim_tol............... 1.05e-08    linesearch_tol.......... 9.00e-01
step_max................ 1.00e+05    max_iter................       50
print_level.........Nag_Soln_Iter    deriv_check.............     TRUE
outfile.................   stdout

Memory allocation:
s.......................      Nag
v.......................      Nag    tdv.....................        3

Results from e04gbc:
--------------------

Iteration results:

  Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step
   0     1    1.0210e+01  3.2e+01  1.9e+00
   1     2    1.9873e-01  2.8e+00  2.4e+00       7.2e-01        1.0e+00
   2     3    9.2324e-03  1.9e-01  2.6e+00       2.5e-01        1.0e+00
   3     4    8.2149e-03  1.2e-03  2.6e+00       2.7e-02        1.0e+00
   4     5    8.2149e-03  5.0e-08  2.6e+00       3.8e-04        1.0e+00
   5     6    8.2149e-03  4.7e-09  2.6e+00       3.6e-06        1.0e+00
   6     7    8.2149e-03  1.2e-09  2.6e+00       6.3e-07        1.0e+00

Final solution:

      x              g            Residuals
  8.24106e-02     1.1994e-09     -5.8811e-03
  1.13304e+00    -1.8647e-11     -2.6535e-04
  2.34370e+00     1.8073e-11      2.7469e-04
                                  6.5415e-03
                                 -8.2299e-04
                                 -1.2995e-03
                                 -4.4631e-03
                                 -1.9963e-02
                                  8.2216e-02
                                 -1.8212e-02
                                 -1.4811e-02
                                 -1.4710e-02
                                 -1.1208e-02
                                 -4.2040e-03
                                  6.8079e-03

The sum of squares is  8.2149e-03.
```

7. Optional Parameters

A number of optional input and output parameters to nag_opt_lsq_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_lsq_deriv; the default settings will then be used for all parameters.

Before assigning values to **options** directly the structure **must** be initialised by a call to the function nag_opt_init (e04xxc). 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 initialisation 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 initialisation.

If assignment of functions and memory to pointers in the **options** structure is required, this must be done directly in the calling program. They cannot be assigned 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_lsq_deriv together with their default values where relevant. The number ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

`Boolean list`	**TRUE**
`Nag_PrintType print_level`	**Nag_Soln_Iter**
`char outfile[80]`	`stdout`
`void (*print_fun)()`	NULL
`Boolean deriv_check`	**TRUE**
`Integer max_iter`	max(50, 5**n**)
`double optim_tol`	$\sqrt{\epsilon}$
`Nag_LinFun minlin`	**Nag_Lin_Deriv**
`double linesearch_tol`	0.9 (0.0 if n = 1)
`double step_max`	100000.0
`double *s`	size **n**
`double *v`	size **n*n**
`Integer tdv`	**n**
`Integer grade`	
`Integer iter`	
`Integer nf`	

7.2. Description of Optional Parameters

list – Boolean Default = **TRUE**

Input: if **options.list** = **TRUE** the parameter settings in the call to nag_opt_lsq_deriv will be printed.

print_level –**Nag_PrintType** Default = **Nag_Soln_Iter**

Input: the level of results printout produced by nag_opt_lsq_deriv. The following values are available.

Nag_NoPrint	No output.
Nag_Soln	The final solution.
Nag_Iter	One line of output for each iteration.
Nag_Soln_Iter	The final solution and one line of output for each iteration.
Nag_Soln_Iter_Full	The final solution and detailed printout at each iteration.

Details of each level of results printout are described in Section 7.3.
Constraint: **options.print_level** = **Nag_NoPrint** or **Nag_Soln** or **Nag_Iter** or **Nag_Soln_Iter** or **Nag_Soln_Iter_Full**.

outfile – char[80] Default = `stdout`

Input: the name of the file to which results should be printed. If **options.outfile**[0] = '\0' 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.

deriv_check – Boolean Default = **TRUE**

Input: if **options.deriv_check** = **TRUE** a check of the derivatives defined by **lsqfun** will be made at the starting point **x**. The derivative check is carried out by a call to nag_opt_lsq_check_deriv (e04yac). A starting point of $x = 0$ or $x = 1$ should be avoided if this test is to be meaningful, but if either of these starting points is necessary then nag_opt_lsq_check_deriv (e04yac) should be used to check **lsqfun** at a different point prior to calling nag_opt_lsq_deriv.

max_iter – Integer Default = max(50, 5**n**)

Input: the limit on the number of iterations allowed before termination.
Constraint: **options.max_iter** ≥ 0.

optim_tol – double Default = $\sqrt{\epsilon}$

Input: the accuracy in x to which the solution is required.
If x_{true} is the true value of x at the minimum, then x_{sol}, the estimated position prior to a normal exit, is such that

$$||x_{sol} - x_{true}|| < \textbf{optim\_tol} \times (1.0 + ||x_{true}||),$$

where $||y|| = \sqrt{\sum_{j=1}^{n} y_j^2}$. For example, if the elements of x_{sol} are not much larger than 1.0 in modulus and if **optim_tol** $= 1.0 \times 10^{-5}$, then x_{sol} is usually accurate to about 5 decimal places. (For further details see Section 9.)
If $F(x)$ and the variables are scaled roughly as described in Section 9and ϵ is the ***machine precision***, then a setting of order **optim_tol** $= \sqrt{\epsilon}$ will usually be appropriate.
Constraint: $10\epsilon \leq$ **options.optim_tol** < 1.0.

minlin – **Nag_LinFun** Default = **Nag_Lin_Deriv**

Input: **minlin** specifies whether the linear minimizations (i.e., minimizations of $F(x^{(k)} + \alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$) are to be performed by a function which just requires the evaluation of the $f_i(x)$, **Nag_Lin_NoDeriv**, or by a function which also requires the first derivatives of the $f_i(x)$, **Nag_Lin_Deriv**.

It will often be possible to evaluate the first derivatives of the residuals in about the same amount of computer time that is required for the evaluation of the residuals themselves – if this is so then nag_opt_lsq_deriv should be called with **minlin** set to **Nag_Lin_Deriv**. However, if the evaluation of the derivatives takes more than about 4 times as long as the evaluation of the residuals, then a setting of **Nag_Lin_NoDeriv** will usually be preferable. If in doubt, use the default setting **Nag_Lin_Deriv** as it is slightly more robust.
Constraint: **options.minlin** = **Nag_Lin_Deriv** or **Nag_Lin_NoDeriv**.

linesearch_tol – double Default = 0.9. (If **n** = 1, default = 0.0)

If **options.minlin** is set to **Nag_Lin_NoDeriv** then the default value of **linesearch_tol** will be changed from 0.9 to 0.5 if **n** > 1.

Input: **linesearch_tol** specifies how accurately the linear minimizations are to be performed.

Every iteration of nag_opt_lsq_deriv involves a linear minimization i.e., minimization of $F(x^{(k)}+\alpha^{(k)}p^{(k)})$ with respect to $\alpha^{(k)}$. The minimum with respect to $\alpha^{(k)}$ will be located more accurately for small values of **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_lsq_deriv, they will increase the number of calls of **lsqfun** made each iteration. On balance it is usually more efficient to perform a low accuracy minimization.
Constraint: $0.0 \leq$ **options.linesearch_tol** < 1.0.

step_max – double Default = 100000.0

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_lsq_deriv will ensure that, for each iteration,

$$\sum_{j=1}^{n} (x_j^{(k)} - x_j^{(k-1)})^2 \leq (\textbf{step\_max})^2$$

where k is the iteration number. Thus, if the problem has more than one solution, nag_opt_lsq_deriv is most likely to find the one nearest to the starting point. On difficult problems, a realistic choice can prevent the sequence $x^{(k)}$ entering a region where the problem is ill-behaved and can help avoid overflow in the evaluation of $F(x)$. However, an underestimate of **options.step_max** can lead to inefficiency.
Constraint: **options.step_max** $\geq$ **options.optim_tol**.

s – double * Default memory = **n**

Input: **n** values of memory will be automatically allocated by nag_opt_lsq_deriv and this is the recommended method of use of **options.s**. However a user may supply memory from the calling program.
Output: the singular values of the Jacobian matrix at the final point. Thus **options.s** may be useful as information about the structure of the user's problem.

v – double * Default memory = **n*n**

Input: **n*n** values of memory will be automatically allocated by nag_opt_lsq_deriv and this is the recommended method of use of **options.v**. However a user may supply memory from the calling program.
Output: the matrix V associated with the singular value decomposition

$$J = USV^T$$

of the Jacobian matrix at the final point, stored by rows. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalised eigenvectors of J^TJ.

tdv – Integer Default = **n**

Input: if memory is supplied by the user then **options.tdv** must contain the last dimension of the array assigned to **options.tdv** as declared in the function from which nag_opt_lsq_deriv is called.
Output: the trailing dimension used by **options.v**. If the NAG default memory allocation has been used this value will be **n**.
Constraint: **options.tdv** $\geq$ **n**.

grade – Integer

Output: the grade of the Jacobian at the final point. nag_opt_lsq_deriv estimates the dimension of the subspace for which the Jacobian matrix can be used as a valid approximation to the curvature (see Gill and Murray, 1978); this estimate is called the grade.

iter – Integer

Output: the number of iterations which have been performed in nag_opt_lsq_deriv.

nf – Integer

Output: the number of times the residuals have been evaluated (i.e., the number of calls of **lsqfun**).

7.3. Description of Printed Output

The level of printed output can be controlled with the structure members **options.list** and **options.print_level** (see Section 7.2.).If **list** = **TRUE** then the parameter values to nag_opt_lsq_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 results printout available from nag_opt_lsq_deriv.

When **options.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:

`Itn`	the current iteration number k.
`Nfun`	the cumulative number of calls to **lsqfun**.
`Objective`	the value of the objective function, $F(x^{(k)})$.
`Norm g`	the Euclidean norm of the gradient of $F(x^{(k)})$.
`Norm x`	the Euclidean norm of $x^{(k)}$.
`Norm(x(k-1)-x(k))`	the Euclidean norm of $x^{(k-1)} - x^{(k)}$.
`Step`	the step $\alpha^{(k)}$ taken along the computed search direction $p^{(k)}$.

When **options.print_level** = **Nag_Soln_Iter_Full** more detailed results are given at each iteration. Additional values output are

`Grade`	the grade of the Jacobian matrix. (See description of **grade**, Section 7.2)
`x`	the current point $x^{(k)}$.
`g`	the current gradient of $F(x^{(k)})$.

`Singular values` the singular values of the current approximation to the Jacobian matrix.

If **options.print_level** = **Nag_Soln** or **Nag_Soln_Iter** or **Nag_Soln_Iter_Full** the final result consists of:

`x`	the final point x^*.
`g`	the gradient of F at the final point.
`Residuals`	the values of the residuals f_i at the final point.
`Sum of squares`	the value of $F(x^*)$, the sum of squares of the residuals at the final point.

If **options.print_level** = **Nag_NoPrint** then printout will be suppressed; the user can print the final solution when nag_opt_lsq_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 **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 **options.print_fun** this will be called in preference to the internal print function of nag_opt_lsq_deriv. Calls to the user defined function are again controlled by means of the **options.print_level** member. Information is provided through **st** and **comm**, the two structure arguments to **print_fun**. If **comm->it_prt = TRUE** then the results from the last iteration of nag_opt_lsq_deriv are in the following members of **st**:

m – Integer
the number of residuals.

n – Integer
the number of variables.

x – double *
points to the **n** memory locations holding the current point $x^{(k)}$.

fvec – double *
points to the **m** memory locations holding the values of the residuals f_i at the current point $x^{(k)}$.

fjac – double *
points to **m*st->tdj** memory locations. **fjac**$[(i-1)$***st->tdj** $+ (j-1)]$ contains the value of $\dfrac{\partial f_i}{\partial x_j}$, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$ at the current point $x^{(k)}$.

tdj – Integer
the trailing dimension for **st->fjac**[ ].

step – double
the step $\alpha^{(k)}$ taken along the search direction $p^{(k)}$.

xk_norm – double
the Euclidean norm of $x^{(k-1)} - x^{(k)}$.

g – double *
points to the **n** memory locations holding the gradient of F at the current point $x^{(k)}$.

grade – Integer
the grade of the Jacobian matrix.

s – double *
points to the **n** memory locations holding the singular values of the current Jacobian.

iter – Integer
the number of iterations, k, performed by nag_opt_lsq_deriv.

nf – Integer
the cumulative number of calls made to **lsqfun**.

The relevant members of the structure **comm** are:

it_prt – Boolean
will be **TRUE** when the print function is called with the result of the current iteration.

sol_prt – 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_lsq_deriv or during a call to **lsqfun** 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 **lsqfun**. 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_2_INT_ARG_LT
On entry, **m** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **m** $\geq$ **n**.
On entry, **tdj** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **tdj** $\geq$ **n**.

NE_DERIV_ERRORS
Large errors were found in the derivatives of the objective function.

The user should check carefully the derivation and programming of expressions for the $\partial f_i/\partial x_j$, because it is very unlikely that **lsqfun** is calculating them correctly.

NE_OPT_NOT_INIT
Options structure not initialised.

NE_BAD_PARAM
On entry parameter **options.print_level** had an illegal value.
On entry parameter **options.minlin** had an illegal value.

NE_2_INT_ARG_LT
On entry, **options.tdv** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **tdv** $\geq$ **n**.

NE_2_REAL_ARG_LT
On entry, **options.step_max** = ⟨*value*⟩ while **options.optim_tol** = ⟨*value*⟩. These parameters must satisfy **step_max** $\geq$ **optim_tol**.

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

NE_INVALID_REAL_RANGE_EF
Value ⟨*value*⟩ given to **options.optim_tol** not valid. Correct range is ⟨*value*⟩$\leq$ **optim_tol** < 1.0.

NE_INVALID_REAL_RANGE_FF
Value ⟨*value*⟩ given to **options.linesearch_tol** not valid. Correct range is $0.0 \leq$ **linesearch_tol** < 1.0.

NE_ALLOC_FAIL
Memory allocation failed.

If one of the above exits occurs, no values will have been assigned to **fsumsq**, or to the elements of **fvec**, **fjac**, **options.s** or **options.v**.

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

If steady reductions in the sum of squares, $F(x)$, were monitored up to the point where this exit occurred, then the exit probably occurred simply because **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.

This could be because **options.optim_tol** has been set so small that rounding errors in the evaluation of the residuals make attainment of the convergence conditions impossible.

NE_SVD_FAIL
The computation of the singular value decomposition of the Jacobian matrix has failed to converge in a reasonable number of sub-iterations.

It may be worth applying nag_opt_lsq_deriv again starting with an initial approximation which is not too close to the point at which the failure occurred.

The exits **NW_TOO_MANY_ITER**, **NW_COND_MIN**, and **NE_SVD_FAIL** may also be caused by mistakes in **lsqfun**, 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.

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 ⟨*string*⟩.

9. Further Comments

The number of iterations required depends on the number of variables, the number of residuals, the behaviour of $F(x)$, the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed per iteration of nag_opt_lsq_deriv varies, but for $m >> n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least one call of **lsqfun**. So, unless the residuals can be evaluated very quickly, the run time will be dominated by the time spent in **lsqfun**.

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 of $F(x)$ 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 nag_opt_lsq_deriv will take less computer time.

When the sum of squares represents the goodness of fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to nag_opt_lsq_covariance (e04ycc), using information returned in the arrays **options.s** and **options.v**. See nag_opt_lsq_covariance (e04ycc) for further details.

9.1. Accuracy

A successful exit (**fail.code** = **NE_NOERROR**) is made from nag_opt_lsq_deriv when (B1, B2 and B3) or B4 or B5 hold, where

$$
\begin{aligned}
B1 &\equiv \alpha^{(k)} \times \| p^{(k)} \| < (\textbf{optim\_tol} + \epsilon) \times (1.0 + \| x^{(k)} \|) \\
B2 &\equiv |F^{(k)} - F^{(k-1)}| < (\textbf{optim\_tol} + \epsilon)^2 \times (1.0 + F^{(k)}) \\
B3 &\equiv \| g^{(k)} \| < \epsilon^{1/3} \times (1.0 + F^{(k)}) \\
B4 &\equiv F^{(k)} < \epsilon^2 \\
B5 &\equiv \| g^{(k)} \| < (\epsilon \times \sqrt{F^{(k)}})^{1/2}
\end{aligned}
$$

and where $||.||$, ϵ and the optional parameter **optim_tol** are as defined in Section 7.2, while $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of first derivatives at $x^{(k)}$.

If **fail.code** = **NE_NOERROR** then the vector in **x** on exit, x_{sol}, is almost certainly an estimate of x_{true}, the position of the minimum to the accuracy specified by **options.optim_tol**.

If **fail.code** = **NW_COND_MIN**, then x_{sol} may still be a good estimate of x_{true}, but to verify this the user should make the following checks. If

(a) the sequence $\{F(x^{(k)})\}$ converges to $F(x_{sol})$ at a superlinear or a fast linear rate, and
(b) $g(x_{sol})^T g(x_{sol}) < 10\epsilon$,

where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum. When (b) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$.

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

10. References

Gill P E and Murray W (1978) Algorithms for the Solution of the Nonlinear Least-squares Problem *SIAM J. Numer. Anal.* **15** 977–992.

11. See Also

nag_opt_lsq_covariance (e04ycc)
nag_opt_init (e04xxc)
nag_opt_read (e04xyc)
nag_opt_free (e04xzc)
nag_opt_lsq_check_deriv (e04yac)

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 **options** structure is declared and initialised by nag_opt_init (e04xxc), a value is then assigned directly to option **optim_tol** and two further options are read from the data file by use of nag_opt_read (e04xyc). The memory freeing function 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 `free()` for this purpose.

12.1. Program Text

```
static void ex2()
{
  double fjac[MMAX][NMAX], fvec[MMAX], x[NMAX];
  Integer i, j, m, n, nt, tdj;
  double fsumsq;
  Boolean print;
  Nag_E04_Opt options;
  Nag_Comm comm;
  static NagError fail, fail2;
  struct user s;

  Vprintf("\n\ne04gbc example 2: using option setting.\n");
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  n = 3;
  m = 15;
  tdj = NMAX;
  nt = 3;

  /* Read data into structure.
   * Observations t (j = 0, 1, 2) are held in s->t[i][j]
   * (i = 0, 1, 2, . . . , 14)
   */
  nt = 3;
  for (i = 0; i < m; ++i)
    {
      Vscanf("%lf", &s.y[i]);
      for (j = 0; j < nt; ++j) Vscanf("%lf", &s.t[i][j]);
    }

  /* Set up the starting point */
  x[0] = 0.5;
  x[1] = 1.0;
  x[2] = 1.5;

  /* Read option values from file */
  fail.print = TRUE;
  print = TRUE;
  e04xyc("e04gbc", "stdin", &options, print, "stdout", &fail);
  if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);

  /* Assign address of user defined structure to
   * comm.p for communication to lsqfun2().
   */
  comm.p = (Pointer)&s;

  /* Call the optimization routine */
  e04gbc(m, n, lsqfun2, x, &fsumsq, fvec, (double *)fjac, tdj,
         &options, &comm, &fail);

  /* Free memory allocated by e04gbc to pointers s and v */
  fail2.print = TRUE;
  e04xzc(&options, "all", &fail2);
  if ((fail.code != NE_NOERROR && fail.code != NW_COND_MIN)
      || fail2.code != NE_NOERROR) exit(EXIT_FAILURE);
} /* ex2 */

#ifdef NAG_PROTO
static void lsqfun2(Integer m, Integer n, double x[], double fvec[],
                    double fjac[], Integer tdj, Nag_Comm *comm)
#else
     static void lsqfun2(m, n, x, fvec, fjac, tdj, comm)
     Integer m, n;
     double x[], fvec[], fjac[];
     Integer tdj;
     Nag_Comm *comm;
#endif
```

```
{
  /* Function to evaluate the residuals and their 1st derivatives
   * in example 2.
   *
   * This function is also suitable for use when Nag_Lin_NoDeriv is specified
   * for linear minimization instead of the default of Nag_Lin_Deriv,
   * since it can deal with comm->flag = 0 or 1 as well as comm->flag = 2.
   *
   * To avoid the use of a global varibale this example assigns the address
   * of a user defined structure to comm.p in the main program (where the
   * data was also read in).
   * The address of this structure is recovered in each call to lsqfun2()
   * from comm->p and the structure used in the calculation of the residuals.
   */

#define FJAC(I,J) fjac[(I)*tdj + (J)]

  Integer i;
  double denom, dummy;
  struct user *s = (struct user *)comm->p;

  for (i = 0; i < m; ++i)
    {
      denom = x[1]*s->t[i][1] + x[2]*s->t[i][2];
      if (comm->flag != 1)
        fvec[i] = x[0] + s->t[i][0]/denom - s->y[i];
      if (comm->flag != 0)
        {
          FJAC(i,0) = 1.0;
          dummy = -1.0 / (denom * denom);
          FJAC(i,1) = s->t[i][0]*s->t[i][1]*dummy;
          FJAC(i,2) = s->t[i][0]*s->t[i][2]*dummy;
        }
    }
}                               /* lsqfun2 */
```

12.2. Program Data

```
Example data for ex2: using option setting
 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0

Following optional parameter settings are read by e04xyc

begin e04gbc

print_level = Nag_Soln_Iter_Full   /* Results printout set to fullest detail */

/* Estimate minimum will be within 10 units of the
 * starting point.
 */
step_max = 10.0

optim_tol = 1.0e-06  /* Set required accuracy of solution */

end
```

12.3. Program Results

```
e04gbc example 2: using option setting.

Optional parameter setting for e04gbc.
--------------------------------------

Option file: stdin

print_level set to Nag_Soln_Iter_Full
step_max set to 1.00e+01
optim_tol set to 1.00e-06

Parameters to e04gbc
--------------------

Number of residuals.......... 15    Number of variables..........  3

minlin..............Nag_Lin_Deriv    machine precision....... 1.11e-16
optim_tol.............. 1.00e-06    linesearch_tol.......... 9.00e-01
step_max............... 1.00e+01    max_iter................       50
print_level....Nag_Soln_Iter_Full    deriv_check.............     TRUE
outfile.................   stdout

Memory allocation:
s.......................      Nag
v.......................      Nag    tdv.....................        3

Results from e04gbc:
--------------------

Iteration results:

  Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
   0     1    1.0210e+01  3.2e+01  1.9e+00                                 3

      x                  g            Singular values
  5.00000e-01         2.1202e+01         4.9542e+00
  1.00000e+00        -1.6838e+01         2.5672e+00
  1.50000e+00        -1.6353e+01         9.6486e-02

  Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
   1     2    1.9873e-01  2.8e+00  2.4e+00       7.2e-01        1.0e+00    3

      x                  g            Singular values
  8.24763e-02         1.8825e+00         4.1973e+00
  1.13575e+00        -1.5133e+00         1.8396e+00
  2.06664e+00        -1.5073e+00         6.6356e-02

  Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
   2     3    9.2324e-03  1.9e-01  2.6e+00       2.5e-01        1.0e+00    3

      x                  g            Singular values
  8.24402e-02         1.3523e-01         4.1026e+00
  1.13805e+00        -9.4890e-02         1.6131e+00
  2.31707e+00        -9.4630e-02         6.1372e-02

  Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
   3     4    8.2149e-03  1.2e-03  2.6e+00       2.7e-02        1.0e+00    3

      x                  g            Singular values
  8.24150e-02         8.1961e-04         4.0965e+00
  1.13323e+00        -5.7539e-04         1.5951e+00
  2.34337e+00        -5.7660e-04         6.1250e-02
```

```
 Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
  4     5    8.2149e-03  5.0e-08  2.6e+00       3.8e-04        1.0e+00    2

      x                 g           Singular values
 8.24107e-02         3.4234e-08        4.0965e+00
 1.13304e+00         8.8965e-09        1.5950e+00
 2.34369e+00        -3.4761e-08        6.1258e-02

 Itn  Nfun   Objective   Norm g   Norm x   Norm (x(k-1)-x(k))   Step    Grade
  5     6    8.2149e-03  4.7e-09  2.6e+00       3.6e-06        1.0e+00    2

      x                 g           Singular values
 8.24106e-02         9.5237e-11        4.0965e+00
 1.13304e+00         3.4598e-09        1.5950e+00
 2.34369e+00        -3.1752e-09        6.1258e-02

Final solution:

      x               g            Residuals
 8.24106e-02      9.5237e-11      -5.8811e-03
 1.13304e+00      3.4598e-09      -2.6536e-04
 2.34369e+00     -3.1752e-09       2.7468e-04
                                   6.5415e-03
                                  -8.2300e-04
                                  -1.2995e-03
                                  -4.4631e-03
                                  -1.9963e-02
                                   8.2216e-02
                                  -1.8212e-02
                                  -1.4811e-02
                                  -1.4710e-02
                                  -1.1208e-02
                                  -4.2040e-03
                                   6.8078e-03

The sum of squares is  8.2149e-03.
```