NAG Toolbox

nag_opt_lsq_uncon_mod_deriv_comp (e04gd)

1 Purpose

nag_opt_lsq_uncon_mod_deriv_comp (e04gd) is a comprehensive modified Gauss–Newton 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 is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Syntax

```
[x, fsumsq, fvec, fjac, s, v, niter, nf, user, ifail] =
nag_opt_lsq_uncon_mod_deriv_comp(m, lsqfun, lsqmon, xtol, x, 'n', n, 'iprint',
iprint, 'maxcal', maxcal, 'eta', eta, 'stepmx', stepmx, 'user', user)
[x, fsumsq, fvec, fjac, s, v, niter, nf, user, ifail] = e04gd(m, lsqfun, lsqmon,
xtol, x, 'n', n, 'iprint', iprint, 'maxcal', maxcal, 'eta', eta, 'stepmx',
stepmx, 'user', user)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 24: **maxcal** was made optional; w and **iw** were removed from the interface; **user** was added to the interface

At Mark 22: *liw* and *lw* were removed from the interface.

3 Description

nag_opt_lsq_uncon_mod_deriv_comp (e04gd) is essentially identical to the function LSQFDN in the NPL Algorithms Library. It is applicable to problems of the form

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

where $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.)

You must supply a function to calculate the values of the $f_i(x)$ and their first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x.

From a starting point $x^{(1)}$ supplied by you, the function 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 $F\left(x^{(k)} + \alpha^{(k)} p^{(k)}\right)$ 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 finite difference estimates of the second derivatives of the $f_i(x)$ are taken into account.

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 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

5 Parameters

5.1 Compulsory Input Parameters

1: **m** – INTEGER

The number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathbf{n} \leq \mathbf{m}$.

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

lsqfun must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. (However, if you do not wish to calculate the residuals or first derivatives at a particular x, there is the option of setting a argument to cause nag_opt_lsq_uncon_mod_deriv_comp (e04gd) to terminate immediately.)

```
[iflag, fvec, fjac, user] = lsqfun(iflag, m, n, xc, ldfjac, user)
```

Input Parameters

1: **iflag** – INTEGER

To **lsqfun**, **iflag** will be set to 1 or 2.

iflag $= 1$
Indicates that only the Jacobian matrix needs to be evaluated

iflag $= 2$
Indicates that both the residuals and the Jacobian matrix must be calculated

2: **m** – INTEGER

m, the numbers of residuals.

3: **n** – INTEGER

n, the numbers of variables.

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

The point x at which the values of the f_i and the $\dfrac{\partial f_i}{\partial x_j}$ are required.

5: **ldfjac** – INTEGER

The first dimension of the array **fjac**.

6: **user** – INTEGER array

lsqfun is called from nag_opt_lsq_uncon_mod_deriv_comp (e04gd) with the object supplied to nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

Output Parameters

1: **iflag** – INTEGER

If it is not possible to evaluate the $f_i(x)$ or their first derivatives at the point given in **xc** (or if it wished to stop the calculations for any other reason), you should reset **iflag** to some negative number and return control to nag_opt_lsq_uncon_mod_deriv_comp (e04gd). nag_opt_lsq_uncon_mod_deriv_comp (e04gd) will then terminate immediately, with **ifail** set to your setting of **iflag**.

2: **fvec**(**m**) – REAL (KIND=nag_wp) array

Unless **iflag** $= 1$ on entry, or **iflag** is reset to a negative number, then **fvec**(i) must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

3: **fjac**($ldfjac$, **n**) – REAL (KIND=nag_wp) array

Unless **iflag** is reset to a negative number, **fjac**(i, j) must contain the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

4: **user** – INTEGER array

Note: **lsqfun** should be tested separately before being used in conjunction with nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

3: **lsqmon** – SUBROUTINE, supplied by the NAG Library or the user.

If **iprint** ≥ 0, you must supply **lsqmon** which is suitable for monitoring the minimization process. **lsqmon** must not change the values of any of its arguments.

If **iprint** < 0, the string `nag_opt_lsq_dummy_lsqmon (e04fdz)` can be used as **lsqmon**.

```
[user] = lsqmon(m, n, xc, fvec, fjac, ldfjac, s, igrade, niter, nf, user)
```

Input Parameters

1: **m** – INTEGER

m, the numbers of residuals.

2: **n** – INTEGER

n, the numbers of variables.

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

The coordinates of the current point x.

4: **fvec**(**m**) – REAL (KIND=nag_wp) array

The values of the residuals f_i at the current point x.

5: **fjac**($ldfjac$, **n**) – REAL (KIND=nag_wp) array

fjac(i, j) contains the value of $\dfrac{\partial f_i}{\partial x_j}$ at the current point x, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

6: **ldfjac** – INTEGER

The first dimension of the array **fjac**.

7: **s**(**n**) – REAL (KIND=nag_wp) array

The singular values of the current Jacobian matrix. Thus **s** may be useful as information about the structure of your problem. (If **iprint** > 0, **lsqmon** is called at the initial point before the singular values have been calculated. So the elements of **s** are set to zero for the first call of **lsqmon**.)

8: **igrade** – INTEGER

nag_opt_lsq_uncon_mod_deriv_comp (e04gd) 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 of the Jacobian matrix, and **igrade** gives its current value.

9: **niter** – INTEGER

The number of iterations which have been performed in nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

10: **nf** – INTEGER

The number of times that **lsqfun** has been called so far with **iflag** $= 2$. (In addition to these calls monitored by **nf**, **lsqfun** is called not more than **n** times per iteration with **iflag** set to 1.)

11: **user** – INTEGER array

lsqmon is called from nag_opt_lsq_uncon_mod_deriv_comp (e04gd) with the object supplied to nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

Output Parameters

1: **user** – INTEGER array

Note: you should normally print the sum of squares of residuals, so as to be able to examine the sequence of values of $F(x)$ mentioned in Section 7. It is usually also helpful to print **xc**, the gradient of the sum of squares, **niter** and **nf**.

4: **xtol** – REAL (KIND=nag_wp)

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 before a normal exit, is such that

$$\|x_{\text{sol}} - x_{\text{true}}\| < \textbf{xtol} \times (1.0 + \|x_{\text{true}}\|)$$

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 **xtol** $= 1.0\text{e}{-5}$, then x_{sol} is usually accurate to about five decimal places. (For further details see Section 7.)

If $F(x)$ and the variables are scaled roughly as described in Section 9 and ϵ is the ***machine precision***, then a setting of order **xtol** $= \sqrt{\epsilon}$ will usually be appropriate. If **xtol** is set to 0.0 or some positive value less than 10ϵ, nag_opt_lsq_uncon_mod_deriv_comp (e04gd) will use 10ϵ instead of **xtol**, since 10ϵ is probably the smallest reasonable setting.

Constraint: **xtol** ≥ 0.0.

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

x(j) must be set to a guess at the jth component of the position of the minimum, for $j = 1, 2, \ldots, n$.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: For **n**, the dimension of the array **x**.

The number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathbf{n} \leq \mathbf{m}$.

2: **iprint** – INTEGER

Suggested value: **iprint** $= 1$.

Default: 1

The frequency with which **lsqmon** is to be called.

iprint > 0
: **lsqmon** is called once every **iprint** iterations and just before exit from nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

iprint $= 0$
: **lsqmon** is just called at the final point.

iprint < 0
: **lsqmon** is not called at all.

iprint should normally be set to a small positive number.

3: **maxcal** – INTEGER

Default: **maxcal** $= 50 \times n$

Enables you to limit the number of times that **lsqfun** is called by nag_opt_lsq_uncon_mod_deriv_comp (e04gd). There will be an error exit (see Section 6) after **maxcal** evaluations of the residuals (i.e., calls of **lsqfun** with **iflag** set to 2). It should be borne in mind that, in addition to the calls of **lsqfun** which are limited directly by **maxcal**, there will be calls of **lsqfun** (with **iflag** set to 1) to evaluate only first derivatives.

Constraint: **maxcal** ≥ 1.

4: **eta** – REAL (KIND=nag_wp)

Suggested value: **eta** $= 0.5$ (**eta** $= 0.0$ if **n** $= 1$).

Default:

if **n** $= 1$, 0.0;
otherwise 0.5.

Every iteration of nag_opt_lsq_uncon_mod_deriv_comp (e04gd) involves a linear minimization, i.e., minimization of $F\left(x^{(k)} + \alpha^{(k)}p^{(k)}\right)$ with respect to $\alpha^{(k)}$. **eta** specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha^{(k)}$ will be located more accurately for small values of **eta** (say, 0.01) than for large values (say, 0.9).

Although accurate linear minimizations will generally reduce the number of iterations, they will tend to increase the number of calls of **lsqfun** (with **iflag** set to 2) needed for each linear minimization. On balance it is usually efficient to perform a low accuracy linear minimization.

Constraint: $0.0 \leq \mathbf{eta} < 1.0$.

5: **stepmx** – REAL (KIND=nag_wp)

Suggested value: **stepmx** $= 100000.0$.

Default: 100000.0

An estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency, a slight overestimate is preferable.) nag_opt_lsq_uncon_mod_deriv_comp (e04gd) will ensure that, for each iteration,

$$\sum_{j=1}^{n}\left(x_j^{(k)} - x_j^{(k-1)}\right)^2 \leq (\mathbf{stepmx})^2$$

where k is the iteration number. Thus, if the problem has more than one solution, nag_opt_lsq_uncon_mod_deriv_comp (e04gd) is most likely to find the one nearest to 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 help avoid overflow in the evaluation of $F(x)$. However, an underestimate of **stepmx** can lead to inefficiency.

Constraint: **stepmx** $\geq$ **xtol**.

6: **user** – INTEGER array

user is not used by nag_opt_lsq_uncon_mod_deriv_comp (e04gd), but is passed to **lsqfun** and **lsqmon**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

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

The final point $x^{(k)}$. Thus, if **ifail** $= 0$ on exit, **x**(j) is the jth component of the estimated position of the minimum.

2: **fsumsq** – REAL (KIND=nag_wp)

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

3: **fvec**(**m**) – REAL (KIND=nag_wp) array

The value of the residual $f_i(x)$ at the final point given in **x**, for $i = 1, 2, \ldots, m$.

4: **fjac**($ldfjac$, **n**) – REAL (KIND=nag_wp) array

The value of the first derivative $\dfrac{\partial f_i}{\partial x_j}$ evaluated at the final point given in **x**, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

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

The singular values of the Jacobian matrix at the final point. Thus **s** may be useful as information about the structure of your problem.

6: **v**(ldv, **n**) – REAL (KIND=nag_wp) array

The matrix V associated with the singular value decomposition

$$J = USV^{\mathrm{T}}$$

of the Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalized eigenvectors of $J^{\mathrm{T}}J$.

7: **niter** – INTEGER

The number of iterations which have been performed in nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

8: **nf** – INTEGER

The number of times that the residuals have been evaluated (i.e., number of calls of **lsqfun** with **iflag** set to 2).

9: **user** – INTEGER array

10: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Note: nag_opt_lsq_uncon_mod_deriv_comp (e04gd) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail < 0 (*warning*)

A negative value of **ifail** indicates an exit from nag_opt_lsq_uncon_mod_deriv_comp (e04gd) because you have set **iflag** negative in **lsqfun**. The value of **ifail** will be the same as your setting of **iflag**.

ifail $= 1$

On entry, $\mathbf{n} < 1$,
or $\mathbf{m} < \mathbf{n}$,
or **maxcal** < 1,
or **eta** < 0.0,
or **eta** ≥ 1.0,
or **xtol** < 0.0,
or **stepmx** $<$ **xtol**,
or $ldfjac < \mathbf{m}$,
or $ldv < \mathbf{n}$,
or $liw < 1$,
or $lw < 7 \times \mathbf{n} + \mathbf{m} \times \mathbf{n} + 2 \times \mathbf{m} + \mathbf{n} \times \mathbf{n}$ when $\mathbf{n} > 1$,
or $lw < 9 + 3 \times \mathbf{m}$ when $\mathbf{n} = 1$.

When this exit occurs, no values will have been assigned to **fsumsq**, or to the elements of **fvec**, **fjac**, **s** or **v**.

ifail $= 2$

There have been **maxcal** evaluations of the residuals. 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 **maxcal** 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.

ifail $= 3$ (*warning*)

The conditions for a minimum have not all been satisfied, but a lower point could not be found. This could be because **xtol** has been set so small that rounding errors in the evaluation of the residuals and derivatives make attainment of the convergence conditions impossible. See Section 7 for further information.

ifail $= 4$

The method for computing 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_uncon_mod_deriv_comp (e04gd) again starting with an initial approximation which is not too close to the point at which the failure occurred.

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

The values **ifail** $= 2$, 3 or 4 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.

7 Accuracy

A successful exit (**ifail** $= 0$) is made from nag_opt_lsq_uncon_mod_deriv_comp (e04gd) when the matrix of approximate second derivatives of $F(x)$ is positive definite, and when (B1, B2 and B3) or B4 or B5 hold, where

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

and where $\|.\|$ and ϵ are as defined in **xtol**, and $F^{(k)}$ and $g^{(k)}$ are the values of $F(x)$ and its vector of estimated first derivatives at $x^{(k)}$.

If **ifail** $= 0$ 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 **xtol**.

If **ifail** $= 3$, then x_{sol} may still be a good estimate of x_{true}, but to verify this you should make the following checks. If

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

(b) $g(x_{\text{sol}})^{\text{T}} g(x_{\text{sol}}) < 10\epsilon$, 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 $F(x_{\text{sol}})$ is a close approximation to $F(x_{\text{true}})$. The values of $F\left(x^{(k)}\right)$ can be calculated in **lsqmon**, and the vector $g(x_{\text{sol}})$ can be calculated from the contents of **fvec** and **fjac** on exit from nag_opt_lsq_uncon_mod_deriv_comp (e04gd).

Further suggestions about confirmation of a computed solution are given in the E04 Chapter Introduction.

8 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_uncon_mod_deriv_comp (e04gd) varies, but for $m \gg n$ is approximately $n \times m^2 + O\left(n^3\right)$. In addition, each iteration makes at least one call of **lsqfun**. So, unless the residuals and their derivatives 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 you will be able to follow these recommendations very closely, but it is worth trying (by guesswork), as sensible scaling will reduce the difficulty of the minimization problem, so that nag_opt_lsq_uncon_mod_deriv_comp (e04gd) 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_uncon_covariance (e04yc), using information returned in the arrays **s** and **v**. See nag_opt_lsq_uncon_covariance (e04yc) for further details.

9 Example

This example finds 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

Before calling nag_opt_lsq_uncon_mod_deriv_comp (e04gd), the program calls nag_opt_lsq_check_deriv (e04ya) to check **lsqfun**. It uses $(0.5, 1.0, 1.5)$ as the initial guess at the position of the minimum.

9.1 Program Text

```
     function e04gd_example

fprintf('e04gd example results\n\n');

% Model fitting data
m = nag_int(15);
y = [ 0.14, 0.18, 0.22, 0.25, 0.29,...
      0.32, 0.35, 0.39, 0.37, 0.58,...
      0.73, 0.96, 1.34, 2.10, 4.39];
t = [ 1.0, 15.0, 1.0;
      2.0, 14.0, 2.0;
      3.0, 13.0, 3.0;
      4.0, 12.0, 4.0;
      5.0, 11.0, 5.0;
      6.0, 10.0, 6.0;
      7.0,  9.0, 7.0;
      8.0,  8.0, 8.0;
      9.0,  7.0, 7.0;
     10.0,  6.0, 6.0;
     11.0,  5.0, 5.0;
     12.0,  4.0, 4.0;
     13.0,  3.0, 3.0;
     14.0,  2.0, 2.0;
     15.0,  1.0, 1.0];
```

```
% Input parameters
xtol   = sqrt(x02aj());
n = nag_int(3);
x = [0.5; 1; 1.5];
user = {y; t};

[x, fsumsq, fvec, fjac, s, v, niter, nf, user, ifail] = ...
    e04gd( ...
           m, @lsqfun, @lsqmon, xtol, x, 'user', user);

fprintf('Best fit model parameters are:\n');
for i = 1:n
  fprintf('        x_%d = %10.3f\n',i,x(i));
end
fprintf('\nResiduals for observed data:\n');
fprintf('  %8.4f  %8.4f  %8.4f  %8.4f  %8.4f\n',fvec);
fprintf('\nSum of squares of residuals    = %7.4f\n',fsumsq);
fprintf('Number of iterations           = %7d\n',niter);
fprintf('Number of function evaluations = %7d\n',nf);

function [iflag, fvecc, fjacc, user] = lsqfun(iflag, m, n, xc, ljc, user)
  y = user{1};
  t = user{2};

  fvecc = zeros(m, 1);
  fjacc = zeros(ljc, n);

  for i = 1:double(m)
    denom = xc(2)*t(i,2) + xc(3)*t(i,3);
    if (iflag ~= 1)
      fvecc(i) = xc(1) + t(i,1)/denom - y(i);
    end
    if (iflag ~= 0)
      fjacc(i,1) = 1;
      dummy = -1/(denom*denom);
      fjacc(i,2) = t(i,1)*t(i,2)*dummy;
      fjacc(i,3) = t(i,1)*t(i,3)*dummy;
    end
  end

function [user] = ...
    lsqmon(m, n, xc, fvecc, fjacc, ljc, s, igrade, niter, nf, user)
```

9.2 Program Results

```
     e04gd example results

Best fit model parameters are:
        x_1 =      0.082
        x_2 =      1.133
        x_3 =      2.344

Residuals for observed data:
   -0.0059   -0.0003    0.0003    0.0065   -0.0008
   -0.0013   -0.0045   -0.0200    0.0822   -0.0182
   -0.0148   -0.0147   -0.0112   -0.0042    0.0068

Sum of squares of residuals    =  0.0082
Number of iterations           =       5
Number of function evaluations =      10
```