NAG Toolbox for MATLAB

e04gd

1 Purpose

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, iw, w, ifail] = e04gd(m,
lsqfun, lsqmon, maxcal, xtol, x, iw, w, 'n', n, 'iprint', iprint, 'eta',
eta, 'stepmx', stepmx, 'liw', liw, 'lw', lw)
```

3 Description

e04gd is essentially identical to the (sub)program 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 (sub)program to calculate the values of the $f_i(x)$ and their first derivatives $\frac{\partial f_i}{\partial x_j}$ 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 – int32 scalar**

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 – string containing name of m-file**

lsqfun must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ 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 parameter to cause e04gd to terminate immediately.)

Its specification is:

```
[iflag, fvecc, fjacc, iw, w] = lsqfun(iflag, m, n, xc, ljc, iw, liw,
w, lw)
```

Input Parameters

1: **iflag – int32 scalar**

To **lsqfun**, **iflag** will be set to 1 or 2. 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 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 e04gd. e04gd will then terminate immediately, with **ifail** set to your setting of **iflag**.

2: **m – int32 scalar**

3: **n – int32 scalar**

The numbers m and n of residuals and variables, respectively.

4: **xc(n) – double array**

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

5: **ljc – int32 scalar**

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

6: **iw(liw) – int32 array**

7: **liw – int32 scalar**

8: **w(lw) – double array**

9: **lw – int32 scalar**

lsqfun is called with e04gd's parameters **iw**, **liw**, **w**, **lw** as these parameters. They are present so that, when other library functions require the solution of a minimization subproblem, constants needed for the evaluation of residuals can be passed through **iw** and **w**. Similarly, you could pass quantities to **lsqfun** from the segment which calls e04gd by using partitions of **iw** and **w** beyond those used as workspace by e04gd. However, because of the danger of mistakes in partitioning, it is **recommended** that you should pass information to **lsqfun** via global variables and **not use iw or w** at all. In any case you **must not change** the elements of **iw** and **w** used as workspace by e04gd.

Output Parameters

1: **iflag – int32 scalar**

To **lsqfun**, **iflag** will be set to 1 or 2. 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 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 e04gd. e04gd will then terminate immediately, with **ifail** set to your setting of **iflag**.

2: **fvecc(m) – double array**

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

3: **fjacc(ljc,n) – double array**

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

4: **iw(liw) – int32 array**
5: **w(lw) – double array**

lsqfun is called with e04gd's parameters **iw**, **liw**, **w**, **lw** as these parameters. They are present so that, when other library functions require the solution of a minimization subproblem, constants needed for the evaluation of residuals can be passed through **iw** and **w**. Similarly, you could pass quantities to **lsqfun** from the segment which calls e04gd by using partitions of **iw** and **w** beyond those used as workspace by e04gd. However, because of the danger of mistakes in partitioning, it is **recommended** that you should pass information to **lsqfun** via global variables and **not use iw or w** at all. In any case you **must not change** the elements of **iw** and **w** used as workspace by e04gd.

Note: lsqfun should be tested separately before being used in conjunction with e04gd.

3: **lsqmon – string containing name of m-file**

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 parameters.

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

Its specification is:

```
[iw, w] = lsqmon(m, n, xc, fvecc, fjacc, ljc, s, igrade, niter, nf,
iw, liw, w, lw)
```

Input Parameters

1: **m – int32 scalar**
2: **n – int32 scalar**

The numbers m and n of residuals and variables, respectively.

3: **xc(n) – double array**

The co-ordinates of the current point x.

4: **fvecc(m) – double array**

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

5: **fjacc(ljc,n) – double array**

fjacc(i,j) contains the value of $\frac{\partial f_i}{\partial x_j}$ at the current point x, for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$

6: **ljc – int32 scalar**

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

7: **s(n) – double 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 – int32 scalar**

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 – int32 scalar**

The number of iterations which have been performed in e04gd.

10: **nf – int32 scalar**

The number of times that user-supplied (sub)program **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: **iw(liw) – int32 array**
12: **liw – int32 scalar**
13: **w(lw) – double array**
14: **lw – int32 scalar**

As in user-supplied (sub)program **lsqfun**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04gd. They are included in **lsqmon**'s parameter list primarily for when e04gd is called by other library functions.

Output Parameters

1: **iw(liw) – int32 array**
2: **w(lw) – double array**

As in user-supplied (sub)program **lsqfun**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04gd. They are included in **lsqmon**'s parameter list primarily for when e04gd is called by other library functions.

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: **maxcal – int32 scalar**

Enables you to limit the number of times that user-supplied (sub)program **lsqfun** is called by 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.

Suggested value: **maxcal** $= 50 \times n$.

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

Constraint: **maxcal** ≥ 1.

5: **xtol – double scalar**

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** $= 1.0\text{D}{-5}$, 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 8 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, e04gd will use 10ϵ instead of **xtol**, since 10ϵ is probably the smallest reasonable setting.

Constraint: **xtol** ≥ 0.0.

6: **x(n) – double array**

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

7: **iw(liw) – int32 array**

Constraint: **liw** ≥ 1.

8: **w(lw) – double array**

Constraints:

if $\mathbf{n} > 1$, $\mathbf{lw} \geq 7 \times \mathbf{n} + \mathbf{m} \times \mathbf{n} + 2 \times \mathbf{m} + \mathbf{n} \times \mathbf{n}$;
if $\mathbf{n} = 1$, $\mathbf{lw} \geq 9 + 3 \times \mathbf{m}$.

5.2 Optional Input Parameters

1: **n – int32 scalar**

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

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 – int32 scalar**

The frequency with which (sub)program **lsqmon** is to be called.

iprint > 0

(sub)program **lsqmon** is called once every **iprint** iterations and just before exit from e04gd.

iprint $= 0$

(sub)program **lsqmon** is just called at the final point.

iprint < 0

(sub)program **lsqmon** is not called at all.

iprint should normally be set to a small positive number.

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

Default: 1

3: **eta – double scalar**

Every iteration of 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 user-supplied (sub)program **lsqfun** (with **iflag** set to 2) needed for each linear minimization. On balance it is usually efficient to perform a low accuracy linear minimization.

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

Default:

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

Constraint: $0.0 \leq$ **eta** < 1.0.

4: **stepmx – double scalar**

An estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency, a slight overestimate is preferable.) 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, 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.

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

Default: 100000.0

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

5: **liw – int32 scalar**

Default: The dimension of the array **iw**.

Constraint: **liw** ≥ 1.

6: **lw – int32 scalar**

Default: The dimension of the array **w**.

Constraints:

if $\mathbf{n} > 1$, $\mathbf{lw} \geq 7 \times \mathbf{n} + \mathbf{m} \times \mathbf{n} + 2 \times \mathbf{m} + \mathbf{n} \times \mathbf{n}$;
if $\mathbf{n} = 1$, $\mathbf{lw} \geq 9 + 3 \times \mathbf{m}$.

5.3 Input Parameters Omitted from the MATLAB Interface

ldfjac, ldv

5.4 Output Parameters

1: **x(n) – double 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 – double scalar**

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) – double 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) – double array**

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

5: **s(n) – double 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) – double 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 – int32 scalar**

The number of iterations which have been performed in e04gd.

8: **nf – int32 scalar**

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

9: **iw(liw) – int32 array**

10: **w(lw) – double array**

11: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

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

ifail < 0

A negative value of **ifail** indicates an exit from e04gd because you have set **iflag** negative in the user-supplied (sub)program **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	$\mathbf{maxcal} < 1$,
or	$\mathbf{eta} < 0.0$,
or	$\mathbf{eta} \geq 1.0$,
or	$\mathbf{xtol} < 0.0$,
or	$\mathbf{stepmx} < \mathbf{xtol}$,
or	$\mathbf{ldfjac} < \mathbf{m}$,
or	$\mathbf{ldv} < \mathbf{n}$,
or	$\mathbf{liw} < 1$,
or	$\mathbf{lw} < 7 \times \mathbf{n} + \mathbf{m} \times \mathbf{n} + 2 \times \mathbf{m} + \mathbf{n} \times \mathbf{n}$ when $\mathbf{n} > 1$,
or	$\mathbf{lw} < 9 + 3 \times \mathbf{m}$ 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 $= 3$

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.

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 e04gd again starting with an initial approximation which is not too close to the point at which the failure occurred.

The values **ifail** $= 2$, 3 or 4 may also be caused by mistakes in user-supplied (sub)program **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 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

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

and where $\|.\|$ and ϵ are as defined in Section 5, 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 $F(x_{\mathrm{sol}})$ at a superlinear or a fast linear rate, and

(b) $g(x_{\mathrm{sol}})^{\mathrm{T}} g(x_{\mathrm{sol}}) < 10\epsilon$, where T denotes transpose, then it is almost certain that x_{sol} is a close approximation to the minimum.

When is true, then usually $F(x_{\mathrm{sol}})$ is a close approximation to $F(x_{\mathrm{true}})$. The values of $F\left(x^{(k)}\right)$ can be calculated in (sub)program **lsqmon**, and the vector $g(x_{\mathrm{sol}})$ can be calculated from the contents of **fvec** and **fjac** on exit from 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 e04gd varies, but for $m \gg n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least one call of user-supplied (sub)program **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 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 e04yc, using information returned in the arrays **s** and **v**. See e04yc for further details.

9 Example

```
e04gd_lsqfun.m
```

```
function [iflag, fvecc, fjacc] = lsqfun(iflag, m, n, xc, ljc)
  global y t;

  fvecc = zeros(m, 1);
  fjacc = zeros(ljc, n);
  for i = 1: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
```

```
e04gd_lsqmon.m
```

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

```
m = int32(15);
maxcal = int32(150);
xtol = 1.05418557512311e-07;
x = [0.5;
     1;
     1.5];
iw = [int32(0)];
w = zeros(105,1);
global y t;
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];
[xOut, fsumsq, fvec, fjac, s, v, niter, nf, iwOut, wOut, ifail] = ...
    e04gd(m, 'e04gd_lsqfun', 'e04gd_lsqmon', maxcal, xtol, x, iw, w)
```

```
xOut =
    0.0824
    1.1330
    2.3437
fsumsq =
    0.0082
fvec =
   -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
fjac =
    1.0000   -0.0401   -0.0027
    1.0000   -0.0663   -0.0095
    1.0000   -0.0824   -0.0190
    1.0000   -0.0910   -0.0303
    1.0000   -0.0941   -0.0428
    1.0000   -0.0931   -0.0558
    1.0000   -0.0890   -0.0692
    1.0000   -0.0827   -0.0827
    1.0000   -0.1064   -0.1064
    1.0000   -0.1379   -0.1379
    1.0000   -0.1820   -0.1820
    1.0000   -0.2482   -0.2482
    1.0000   -0.3585   -0.3585
    1.0000   -0.5791   -0.5791
    1.0000   -1.2409   -1.2409
s =
    4.0965
    1.5950
    0.0613
v =
    0.9354    0.3530   -0.0214
   -0.2592    0.6432   -0.7205
   -0.2405    0.6795    0.6932
niter =
           5
nf =
          10
iwOut =
           0
wOut =
     array elided
ifail =
           0
```