8.2 Solve System of Nonlinear Equations

A. Purpose

This subroutine solves a system of n nonlinear equations in n unknowns. The problem can be expressed as

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

where $\mathbf{x}$ denotes an n-vector of unknowns and $\mathbf{f}$ denotes an n-dimensional vector-valued function, with components $f_i(\mathbf{x})$, $i = 1, ..., n$.

It is assumed that the function $\mathbf{f}$ has continuous first partial derivatives with respect to $\mathbf{x}$, at least in a reasonably sized neighborhood of the solution.

The solution algorithm makes use of the $n \times n$ Jacobian matrix, $J(\mathbf{x})$, whose (i, j) element is $\partial f_i/\partial x_j$ evaluated at $\mathbf{x}$. The elements of J may either be computed by user-provided code or estimated by the algorithm using finite differences of values of $\mathbf{f}$.

An auxiliary subroutine DCKDER (Chapter 8.3) may be used as an aid for checking the consistency of code the user may write for computing derivatives.

The problem, $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, can also be solved using the nonlinear least-squares package DNLxxx of Chapter 9.3. Generally if a problem can be solved by both DNQSOL and DNLxxx the execution time will be significantly less using DNQSOL. Problems for which DNLxxx is needed are:

(1) those in which the number of unknowns is not equal to the number of equations,

(2) those in which one does not expect the system to have an exact solution,

(3) those in which constraints on the variables are an inherent aspect of the problem and one expects that the solution may lie on a constraint boundary, or

(4) those in which it is necessary to constrain the region of search to prevent the algorithm from wandering away from the expected neighborhood of the solution.

B. Usage

B.1 Program Prototype, Double Precision

EXTERNAL DNQFJ

INTEGER N, IOPT(), IDIMW

DOUBLE PRECISION X(≥N)**, FVEC**(≥N)**, XTOL, W**(IDIMW)

Assign values to N, X(), XTOL, IOPT(), IDIMW, and optionally, W().

CALL DNQSOL(DNQFJ, N, X, FVEC, XTOL, IOPT, W, IDIMW)

Computed results will be returned in X(), FVEC(), IOPT(1:3), and W(3).

B.2 Argument Definitions

DNQFJ [in] Name of subroutine provided by the user to compute the value of $\mathbf{f}$, and optionally J, at any specified value of $\mathbf{x}$. This subroutine must be of the form:

```
subroutine DNQFJ(N, X, FVEC, FJAC, IFLAG)
integer N, IFLAG
double precision X(N), FVEC(N), FJAC(N,N)
On entry X() contains the current value
of x.
If IFLAG = 0, print X() and FVEC().
   {This action is needed only if the
   iprint option is selected with nprint
   positive.}
If IFLAG = 1, compute f(x), storing f_i in
   FVEC(i) for i = 1, ...,N.
If IFLAG = 2, compute J(x), storing
   ∂f_i/∂x_j in FJAC(i,j) for i = 1, ..., N,
   and j = 1, ..., N. {This action is not
   needed if the inoj option is selected,
   however FJAC must still appear in the
   argument list.}
return
end
```

The subroutine DNQFJ may set IFLAG to a negative integer value to cause immediate termination of the solution procedure. Otherwise, the value of IFLAG should not be changed.

Except at the initial $\mathbf{x}$, a call to DNQFJ with IFLAG = 2 will almost never be with the same $\mathbf{x}$ as the previous call with IFLAG = 1. Thus, it is not easy to save subexpressions from the computation of function values for reuse in the computation of the Jacobian. The Jacobian updating method used by this algorithm reduces the number of Jacobian evaluations needed.

In some applications DNQFJ may need additional data that may have been input by the user's main program. One way to handle this in Fortran 77 is by use of named COMMON blocks.

N [in] The number, n, of components in $\mathbf{x}$ and in $\mathbf{f}$.

X() [inout] On entry X() must contain an initial value for the **x** vector. On successful termination X() contains the final estimate of the solution vector, **x**.

FVEC() [out] On termination FVEC() contains the value of the **f** vector evaluated at the final **x**.

XTOL [in] Require XTOL ≥ 0.0. The algorithm will report successful termination when it estimates the relative error in **x**, in a weighted Euclidean vector norm sense, is less than $\max(\text{XTOL}, \epsilon)$, where ϵ is the machine accuracy, given by D1MACH(4) (or R1MACH(4) for single precision). D1MACH and R1MACH are described in Chapter 19.1.

We suggest setting XTOL in the range from $\epsilon^{0.50}$ to $\epsilon^{0.75}$.

See Section D for further discussion of XTOL and the convergence test.

IOPT() [inout] Array used to select options and to return information. The first three locations are used to return information as follows:

IOPT(1) = $info$. Indicator of status on termination. If $info < 0$, termination is due to IFLAG having been set negative in DNQFJ, and $info$ will have the value that was assigned to IFLAG. Otherwise, $info$ may have the following values:

- 0 Successful termination. Either the XTOL test is satisfied or the condition $\mathbf{f} = \mathbf{0}$ is satisfied exactly.
- 1 Improper input parameters. Require N > 0, IDIMW $\geq$ the value specified below, and valid entries in IOPT(i) for $i \geq 4$.
- 2 Number of calls to DNQFJ has reached the limit of *maxfev*. Default: $maxfev = 200 \times (\text{N} + 1)$.
- 3 It appears to be impossible to satisfy the XTOL test. Possibly XTOL should be set larger.
- 4, 5 The value 4 means there have been five successive evaluations of the Jacobian matrix without any significant reduction in $\|\mathbf{f}\|$, while 5 means there have been ten successive evaluations of **f** without any significant reduction in $\|\mathbf{f}\|$. Typically the algorithm will terminate with IOPT(1) = 4 or 5 when it is trapped in the neighborhood of a local minimum of $\|\mathbf{f}\|$ at which **f** is not the zero vector.

When IOPT(1) = 2, 4, or 5 it is advisable to check the validity of the user code, if any, for computing the Jacobian. Once that is assured it may be useful to try different initial values for **x**.

IOPT(2) = $nfev$. Number of function evaluations used, *i.e.*, the number of times DNQFJ was called with IFLAG = 1.

IOPT(3) = $njev$. Number of evaluations of the Jacobian matrix, either by finite-difference approximation or by calling DNQFJ with IFLAG = 2.

Locations in IOPT() indexed from 4 on are available for selecting options. The sequence of option selections must be terminated with a zero. For the simplest usage, if DNQFJ contains code for computing the Jacobian matrix just set IOPT(4) = 0, while if DNQFJ does not contain code for the Jacobian set IOPT(4) = 1 and IOPT(5) = 0.

Options have code numbers from 1 to 8, and some options have one or two integer arguments. The code numbers of options to be exercised, each followed immediately by its arguments, if any, must be placed in IOPT() beginning at IOPT(4), with no gaps, and terminated by a zero value. The ordering of different options in IOPT() is not significant. If an option is repeated the last occurrence prevails.

As a convenience for altering the set of selected options, the negative of an option code is interpreted to mean the option is to take its default value. If this is an option that has arguments, space for the appropriate number of arguments must still be allocated following the option code, even though the arguments will not be used.

The option codes and their arguments are as follows:

- 1 *inoj*. Select this option if the subroutine DNQFJ does not contain code for computing the Jacobian matrix. In this case no calls to DNQFJ will be made with IFLAG = 2, and the algorithm will estimate the Jacobian matrix by computing finite differences of values of **f**.
- 2 *isetd*. Argument: *dmode*. Selects manner of initializing and subsequently altering the weighting array, *diag*(), which is stored in W(4 : N+3) and discussed in Section D. *dmode* may be set to 1, 2, or 3. The default value is 1.
  - 1 DNQSOL will set *diag*() to all ones and not subsequently alter these values. Reference [1] states that, for most tested problems, the MINPACK code from which the present package was derived was most successful using this option.
  - 2 The user must assign positive values to *diag*() before calling DNQSOL. DNQSOL will not subsequently alter these values.
  - 3 DNQSOL will initially set $diag(j)$ to be the Euclidian norm of the j^{th} column of the initial Jacobian matrix, for $j = 1, ..., \text{N}$. If the j^{th} column is all zeros $diag(j)$ will be set to one. The value of $diag(j)$ will be increased

subsequently if the norm of the j^{th} column increases.

3 *iprint*. Argument: *nprint*. The algorithm increments an internal counter, *ibest*, whenever a new value of $\mathbf{x}$ is accepted as giving a significant reduction in $\|\mathbf{f}\|$. If $nprint > 0$, DNQFJ will be called with IFLAG = 0 after the first evaluation of $\mathbf{f}$, after every $nprint^{th}$ time *ibest* is incremented, and on termination. DNQFJ is expected to print X() and FVEC() on these occasions. The values in X() and FVEC() on these calls will be those that have given the smallest value of $\|\mathbf{f}\|$ up to that point. Thus, these will not always be the values associated with the most recent function evaluation. Setting $nprint \leq 0$ has the same effect as not exercising option *iprint*, *i.e.*, no calls will be made to DNQFJ with IFLAG = 0.

4 *imxfev*. Argument: *maxfev*. The algorithm will terminate with $info = 2$ if convergence has not been attained after *maxfev* function evaluations. Default value: $200 \times (\mathrm{N} + 1)$.

5 *iband*. Arguments: *ml* and *mu*. If DNQFJ will not be computing the Jacobian matrix, J, and thus option *inoj* is selected, and if J has a sufficiently narrow band structure, a reduction in the number of function evaluations needed to estimate J by finite differences can be effected by informing the algorithm of the band structure. In such a case set *ml* and *mu* to values such that all the nonzero elements of J lie within the first *ml* subdiagonals, the main diagonal, and the first *mu* superdiagonals. This only reduces the number of function evaluations if $ml + mu + 1 <$ N.

6 *iepsfn*. Select this option if DNQFJ will not be computing the Jacobian matrix, J, and thus option *inoj* is selected, and you are providing an estimate, *epsfcn*, in W(1), of the relative error expected in function evaluations. *epsfcn* is used by the algorithm in selecting the increment used in computing finite difference approximations to derivatives. Default: *epsfcn* = machine precision obtained as D1MACH(4) (or R1MACH(4) in single precision.)

7 $ifactr$. Select this option if you are providing a value, $factor$, in W(2). *factor* provides a bound on the weighted length of the first step the algorithm takes. Default: $factor = 0.75$. See Section D for more information on the role of $factor$.

8 *itrace*. This option activates printing of intermediate diagnostic output. The output statements use unit " * " to direct the output to the standard system output unit.

W() [in, out, scratch] An array of length IDIMW.

If option *iepsfn* is selected the user must store *epsfcn* in W(1). This value will not be altered.

If option $ifactr$ is selected the user must store $factor$ in W(2). This value will not be altered.

On return W(3) will contain a quantity, *toltst*, that can be regarded as an estimate of the relative error in the final $\mathbf{x}$ in a weighted norm sense. If $\mathbf{x} \neq 0$, $toltst = \Delta / \|Dx\|$, which is the value compared with XTOL for the convergence test (see Section D).

If $\mathbf{x} = 0$, $toltst = 0$.

If option *isetd* is selected with *dmode* = 2, the user must store positive values for *diag*(1 : N) in W(4 : N+3). In this case the contents of W(4 : N+3) will not be altered by DNQSOL. If *isetd* is not selected or if it is selected with *dmode* = 1 or 3, values for *diag*() will be stored in W(4 : N+3) by DNQSOL.

W(N+4 : 3+(15×N+3×N^2)/2) is used as working space.

IDIMW [in] Dimension of W() array. Require IDIMW $\geq 3 + (15 \times \mathrm{N} + 3 \times \mathrm{N}^2)/2$.

B.3 Modifications for Single Precision

For single precision usage change the DOUBLE PRECISION statements to REAL and change the initial letters of the subroutine names from "D" to "S." It is recommended that one use the double precision rather than the single precision version of this package for better reliability, except on computers such as the Cray Y/MP that have precision of about 14 decimal places in single precision.

C. Examples and Remarks

Consider the sample problem:

$$\exp(-x_1) + \sinh(2x_2) + \tanh(2x_3) = 5.01$$
$$\exp(2x_1) + \sinh(-x_2) + \tanh(2x_3) = 5.85$$
$$\exp(2x_1) + \sinh(2x_2) + \tanh(-x_3) = 8.88$$

To use DNQSOL this must be put in the form of expressions whose desired values are zeros. Thus, for instance we may define

$$f_1 \equiv \exp(-x_1) + \sinh(2x_2) + \tanh(2x_3) - 5.01 = 0$$
$$f_2 \equiv \exp(2x_1) + \sinh(-x_2) + \tanh(2x_3) - 5.85 = 0$$
$$f_3 \equiv \exp(2x_1) + \sinh(2x_2) + \tanh(-x_3) - 8.88 = 0$$

The program DRDNQSOL illustrates the use of DNQSOL to solve this problem. DRDNQSOL also illustrates

the use of DCKDER of Chapter 8.3 to check the mutual consistency of the code for function and Jacobian evaluation. Results are shown in ODDNQSOL.

A problem $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ may have no solutions, or one or more solutions. The residual norm $\|\mathbf{f}(\mathbf{x})\|$ may have more than one local minimum, some with $\mathbf{f}$ not zero. The norm $\|\mathbf{f}(\mathbf{x})\|$ can have a nonzero minimum only at a point where the Jacobian matrix is singular. The user should choose an initial $\mathbf{x}$ that is as close to the desired solution as possible. For a problem whose properties with regard to multiplicity of solutions or local minima are not known, it may be advisable to apply the subroutine several times using different initial $\mathbf{x}$'s to see if different termination points occur.

It is often useful, or even necessary, to limit the search to a specified region. Subroutine DNQSOL does not have a specific feature for bounding variables. One can limit the size of the initial step by setting $factor$ using option $ifactr$. If bounding of variables is definitely needed, one can use the nonlinear least-squares package of Chapter 9.3. If a problem can be solved by DNQSOL, this approach will generally require less execution time than would the use of the Chapter 9.3 package.

Since the uniformly weighted Euclidean norm of $\mathbf{f}$ plays a central role in the algorithm, it is generally advantageous to have the separate components of $\mathbf{f}$ scaled so that an error of one unit in one component of $\mathbf{f}$ has about the same importance as an error of one unit in any other component.

If the user has estimates of positive numbers dx_j such that a change of magnitude dx_j in x_j is expected to have about the same effect on $\|\mathbf{f}\|$ as a change of magnitude dx_i in x_i for all i and j, it may be useful to set $diag(j) = 1/dx_j$ for $j = 1, ..., n$, by using the option $isetd$. The overall scaling of values for $diag()$ must be coordinated with the choice of XTOL and $factor$ to assure that the convergence test (see end of Section D) makes sense. For example it may be convenient to scale $diag()$ so its largest element is 1.

D. Functional Description

The algorithm attempts to find a vector $\hat{\mathbf{x}}$ satisfying $\mathbf{f}(\hat{\mathbf{x}}) = \mathbf{0}$, starting the search from a given point $\mathbf{x}_0$.

The algorithm is a trust-region method, using a Broyden update to reduce the number of times the Jacobian matrix must be recomputed, and using a double-dogleg method of solving the constrained linearized problem within the trust-region. For detailed descriptions of these terms and techniques, see [2]. A brief description follows:

Given a point, $\mathbf{x}$, a positive diagonal scaling matrix, D, and a trust-region radius, Δ, the algorithm attempts to find a step vector, $\mathbf{p}$, that solves the modified problem:

$$\min\{\|\mathbf{f}(\mathbf{x}+\mathbf{p})\|^2 : \|D\mathbf{p}\| \le \Delta\}$$

where $\|\cdot\|$ denotes the Euclidean norm. The algorithm replaces this problem by the linearization,

$$\min\{\|\mathbf{f}+J\mathbf{p}\|^2 : \|D\mathbf{p}\| \le \Delta\}$$

where $\mathbf{f}$ and J denote the function vector and Jacobian matrix evaluated at $\mathbf{x}$. This problem is approximately solved using a double-dogleg method that further restricts the step vector $\mathbf{p}$ to be in the two-dimensional subspace spanned by the Gauss-Newton direction, $-J^{-1}\mathbf{f}$, and the scaled gradient descent direction, $-D^{-2}J^t\mathbf{f}$.

The region $\{\mathbf{x}+\mathbf{p} : \|D\mathbf{p}\| \le \Delta\}$ is called the trust-region associated with $\mathbf{x}$. The algorithm attempts to regulate the size of Δ so the linear function $\mathbf{f}+J\mathbf{p}$ will be a reasonably good approximation to the nonlinear function $\mathbf{f}(\mathbf{x}+\mathbf{p})$ within the trust-region.

Diagonal elements of D are stored in the array $diag()$, which is initialized, and optionally updated, as described in Section B. The trust-region radius Δ is initially computed as $\Delta = factor \times \|\mathbf{x}_0\|$, if $\|\mathbf{x}_0\| > 0$, and as $\Delta = factor$, otherwise. The initialization of $factor$ is described in Section B.

Let $\mathbf{p}$ denote the step vector determined by this process and define $\mathbf{x}_+ = \mathbf{x}+\mathbf{p}$. If $\|\mathbf{f}(\mathbf{x}_+)\| \ge \|\mathbf{f}(\mathbf{x})\|$ the algorithm will reduce Δ and solve for a new $\mathbf{p}$.

If a smaller value of $\|\mathbf{f}\|$ is found, the algorithm computes an estimate of the Jacobian at the new point and starts the step determination process again. Rather than always computing the new Jacobian by calling the user-supplied code or using finite differences, the algorithm first tries to use an updated Jacobian which is more economical to compute. The update formula, due to Broyden, is

$$J_+ = J + \frac{(\mathbf{f}(\mathbf{x}+\mathbf{p}) - \mathbf{f}(\mathbf{x}) - J\mathbf{p})\,(D^tD\mathbf{p})^t}{\|D\mathbf{p}\|^2}$$

The algorithm computes and maintains a QR factorization of J. The Broyden update process is applied to the QR factorization of J.

If progress is unsatisfactory when based on an updated Jacobian, the algorithm will recompute the Jacobian using the user-supplied code for J, if available, or otherwise by finite differences of user-computed values of $\mathbf{f}$.

As the algorithm makes progress toward a zero of $\mathbf{f}$, the step $\mathbf{p}$ will generally become smaller. If also the trust-region seems reliable, in the sense that the relative reduction in $\|\mathbf{f}\|^2$ is within 10% of the relative reduction predicted by the linear model, then Δ will be reset to

$2\|D\mathbf{p}\|$ so that Δ decreases as $\|D\mathbf{p}\|$ decreases. Convergence to a zero of $\mathbf{f}$ is assumed to have occurred when

$$\Delta \leq \max(\text{XTOL}, \epsilon) \times \|D\mathbf{x}\|,$$

or if an $\mathbf{x}$ is found for which $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, exactly. In these cases the subroutine returns with IOPT(1) = 1.

If the algorithm gets trapped in the neighborhood of a local minimum of $\|\mathbf{f}\|$ at which $\mathbf{f}$ is not zero, the step $\mathbf{p}$, and thus the trust-region radius, Δ, will generally not get small. The Jacobian J will be singular at such a local minimum. These conditions generally result in a return with IOPT(1) = 4 or 5 or possibly 2.

References

1. M. J. D. Powell, *A hybrid method for nonlinear equations*, in P. Rabinowitz, editor, **Numerical Methods for Nonlinear Algebraic Equations**, Gordon and Breach (1970). Harwell library subroutine NS01A, described earlier by Powell in Harwell reports AERE–R–5947 (1968) *A Fortran Subroutine for Solving Systems of Non-linear Algebraic Equations,* and T.P. 364 (1969) *A Hybrid Method for Non-linear Equations.*

2. John E. Dennis Jr. and Robert B. Schnabel, **Numerical Methods for Unconstrained Optimization and Nonlinear Equations**, Prentice-Hall, Englewood Cliffs, N. J. (1983) 378 pages.

3. Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom, **User Guide for MINPACK-1**. Technical Report ANL–80–74, Argonne National Laboratory (1980) 261 pages.

4. Burton S. Garbow, Kenneth E. Hillstrom, and Jorge J. Moré, **Implementation Guide for MINPACK-1**. Technical Report ANL–80–68, Argonne National Laboratory (1980) 58 pages.

E. Error Procedures and Restrictions

Notice of any detected error condition is sent to the error message printing routines of Chapter 19.2 using error level 0, and the subroutine will return with a value of *info* in IOPT(1) other than 0. The action of the error printing routines can be altered as described in Chapter 19.2.

F. Supporting Information

Entry	Required Files
DNQSOL	AMACH, DERV1, DNQSOL, DNRM2, ERFIN, ERMSG, IERM1, IERV1
SNQSOL	AMACH, ERFIN, ERMSG, IERM1, IERV1, SERV1, SNQSOL, SNRM2

The source language is ANSI Fortran 77.

The original MINPACK version of this code was designed and programmed by the authors of [3] and [4] where this code is called HYBRD and HYBRJ. They, in turn, cite [1] for the general strategy of their approach and numerous other authors for additional ideas. The MINPACK code was downloaded from *netlib* to JPL by C. L. Lawson in February 1990. The subroutine names and the user interface were modified by Lawson and F. T. Krogh in 1990 and 1991 to conform to the MATH77 library style. Changes to the logic of the algorithm have been made that reduced the execution time and improved the reliability of the code when applied to the set of fourteen test problems included with the MINPACK package.

DRDNQSOL

```
c      program DRDNQSOL
c>> 1996-06-21 DRDNQSOL Krogh  Changes for C conversion.
c>> 1994-11-02 DRDNQSOL Krogh  Changes to use M77CON
c>> 1992-04-15 DRDNQSOL CLL.
c>> 1992-01-14 CLL.
c      Demo driver for DNQSOL.  Also using DCKDER.
c      Expected solution vector:  0.9000518      1.0001835      1.0945009
c     ------------------------------------------------------------------
c--D replaces "?": DR?NQSOL, ?NQSOL, ?NRM2, ?CKDER, ?NQFJ
c     ------------------------------------------------------------------
      external D1MACH, DNQFJ, DNRM2
      integer I, IMAX, IOPT(5), J, JMAX, LWA, M, MODE, N, NMAX
      parameter(NMAX = 3, LWA = 3+(15*NMAX+3*NMAX*NMAX)/2 )
      double precision D1MACH, DNRM2
      double precision FJAC(NMAX,NMAX), FNORM, FVEC(NMAX)
      double precision TEST(NMAX,NMAX), TOL, TSTMAX, WA(LWA), X(NMAX)
      data N / NMAX /
c     ------------------------------------------------------------------
      IOPT(4) = 0
```

```
      TOL = sqrt(D1MACH(4))
      X(1) = 3.0D0
      X(2) = 3.0D0
      X(3) = 3.0D0
      print*,
     *  'Program DRDNQSOL.  Demo driver for DNQSOL.  Also using DCKDER.'

c     ------------------------------------------------------------------
c                Using DCKDER to check derivative computation.
c     ------------------------------------------------------------------
      print'(/'' Using DCKDER to check derivative computation.'')'
      M  = N
      call DNQFJ(N, X, FVEC ,FJAC, 2)
      MODE = 1
   10 continue
      call DCKDER(MODE, M, N, X, FVEC, FJAC, NMAX, TEST,
     *            IMAX, JMAX, TSTMAX)
      if(MODE .eq. 2) then
         call DNQFJ(N, X, FVEC ,FJAC, 1)
         go to 10
      endif
      call DNQFJ(N, X, FVEC ,FJAC, 1)
      print'(/11x,''X(J) ='',5g11.3:/(17x,5g11.3))',(X(J),J=1,N)
      print'(/1x,''  I     FVEC(I)   '',
     *   ''  .....................FJAC(I,J).......................''/)'
      do 20 I = 1,M
         print'(1x,i3,1x,g11.3,1x,5g11.3:/(17x,5g11.3))',
     *        I,FVEC(I),(FJAC(I,J),J=1,N)
   20 continue
      print'(/1x,''TEST(,):''/)'
      do 30 I = 1,M
         print'(1x,i3,13x,5g11.3:/(17x,5g11.3))',I,(TEST(I,J),J=1,N)
   30 continue
      print'(/1x,''IMAX ='',i3,'',     JMAX ='',i3,'',     TSTMAX ='',
     *   g11.3)', IMAX,JMAX,TSTMAX

c     ------------------------------------------------------------------
c           Using DNQSOL to solve system of nonlinear equations.
c     ------------------------------------------------------------------
      print
     *  '(/'' Using DNQSOL to solve system of nonlinear equations.'')'

      call DNQSOL(DNQFJ, N, X, FVEC, TOL, IOPT, WA, LWA)

      FNORM = DNRM2(N,FVEC,1)
c++ CODE for ~.C. is active
      print'('' Termination status:  '',i6/'' NFEV, NJEV:          '',
     * 2i6/ '' Final residual norm: '',g14.3/'' Final X():          ''
     * /(8x,4f14.7))',  IOPT(1), IOPT(2), IOPT(3),
     * FNORM, (X(J), J = 1, N)
c++ CODE for .C. is inactive
c%%   printf( " Termination status:  %6ld\n NFEV, NJEV:          "
c%%   "%6ld%6ld\n Final residual norm: %14.3g\n Final X():          ",
c%%      Iopt[1], Iopt[2], Iopt[3], fnorm);
c%%   for (j = 0; j < n; j+=4){
c%%      printf( "\n        " );
c%%      for (i = j; i < (j < n - 3 ? j + 4 : n); i++)
c%%         printf( "%14.7f", x[i] );}
c%%   printf( "\n" );
```

```
c++ END
      stop
      end
c     ==================================================================
      subroutine DNQFJ(N, X, FVEC ,FJAC, IFLAG)
c>> 1992-01-14 CLL.
c     Sample 3-dimensional function of 3 variables for demo of solution
c     of a system of nonlinear equations.
c     ------------------------------------------------------------------
      integer I, IFLAG, N
      double precision C1(3), C2(3), C3(3), FJAC(N,N), FVEC(N)
      double precision TERM(3), X(N)
      data C1 / -1.0d0,  2.0d0,  2.0d0 /
      data C2 /  2.0d0, -1.0d0,  2.0d0 /
      data C3 /  2.0d0,  2.0d0, -1.0d0 /
      data TERM / 5.01d0, 5.85d0, 8.88d0 /
c     ------------------------------------------------------------------
      if (IFLAG .eq. 1) then
c                                          Compute function vector.
         do 10 I = 1,N
            FVEC(I) = exp(C1(I)*X(1)) + sinh(C2(I)*X(2)) +
     *                tanh(C3(I)*X(3)) - TERM(I)
   10    continue
      elseif (IFLAG .eq. 2) then
c                                          Compute Jacobian matrix.
         do 40 I = 1, N
            FJAC(I,1) = exp(C1(I)*X(1)) * C1(I)
            FJAC(I,2) = cosh(C2(I)*X(2)) * C2(I)
            FJAC(I,3) = (1.0d0/cosh(C3(I)*X(3)))**2 * C3(I)
   40    continue
      endif
      return
      end
```

ODDNQSOL

```
Program DRDNQSOL.  Demo driver for DNQSOL.  Also using DCKDER.

Using DCKDER to check derivative computation.

          X(J) =   3.00        3.00        3.00

  I    FVEC(I)    ..................FJAC(I,J).....................

  1     198.      -0.498E-01   403.       0.492E-04
  2     389.       807.       -10.1      0.492E-04
  3     595.       807.        403.     -0.987E-02

TEST(,):

  1               0.368E-10  -0.185E-06  0.103E-09
  2              -0.370E-06   0.172E-08  0.103E-09
  3              -0.370E-06  -0.185E-06  0.462E-09

IMAX =  2,    JMAX =  1,    TSTMAX =  0.370E-06

Using DNQSOL to solve system of nonlinear equations.
Termination status:       0
```

```
NFEV, NJEV:                38       5
Final residual norm:        0.516E-09
Final X():
             0.9000518      1.0001835      1.0945009
```