9.3 Nonlinear Least-Squares

A. Purpose

The subroutines in this package estimate parameters in a nonlinear model using the criterion of least-squares. Eight different user-callable subroutines are provided to support different combinations of three options: (1) the method of obtaining derivative values, (2) bounds on parameters, and (3) use of a special algorithm for the separable problem.

We urge you not to be put off by the length and complexity of this documentation. The length is primarily due to the large number of internal parameters that are made accessible via the IV() and V() arrays. For simplest usage you can just set IV(1) = 0 and not store anything into the V() array. Then most of the information needed to use the package will be found in Section B.1 for the nonseparable subroutines or in Section B.2 for the separable subroutines. Two sample main programs are described in Section C and listed following Section F.

A typical application of this software would be the identification of parameters in a nonlinear model to provide the best fit to measured data in the sense of minimizing the sum of squares of differences between values of the model function and the observed data. For example one could define a residual function as

$$r_i(\mathbf{c}) = f(t_i; \mathbf{c}) - y_i, \quad i = 1, ..., NDATA, \tag{1}$$

where y_i is an item of measured data associated with an independent variable value t_i, and f is a model function depending nonlinearly on the NC-dimensional parameter vector $\mathbf{c}$. The objective function that will be minimized by subroutines of this package is

$$\psi(\mathbf{c}) = \frac{1}{2} \sum_{i=1}^{NDATA} r_i(\mathbf{c})^2. \tag{2}$$

The function r_i could be defined in ways other than the expression in Eq. (1). The package does assume that $r_i(\mathbf{c})$ is a continuously differentiable function of $\mathbf{c}$, [1, p. 369].

It is not uncommon for some parameters in a nonlinear least-squares problem to occur linearly. When this is the case we shall say the problem is separable. Algorithms specialized for the separable problem have been developed. By reducing the dimension of the parameter space in which the nonlinear search must be executed, such an algorithm generally has better efficiency and robustness than an algorithm that does not take advantage of the separability.

A problem is separable if there is a way the coefficient vector, $\mathbf{c}$, can be partitioned into two subsets, say α and β, such that, with the α coefficients held at fixed values, the model function is a linear function of the β coefficients. With such a partition we will refer to α as the nonlinear coefficients, and β as the linear coefficients. Suppose α has NA components and β has NB components. The residual function will be regarded as having one of the forms

$$r_i(\alpha, \beta) = \beta_1 \varphi_{i,1}(\alpha) + \cdots + \beta_{NB} \varphi_{i,NB}(\alpha) - y_i, \text{ or} \tag{3}$$

$$r_i(\alpha, \beta) = \beta_1 \varphi_{i,1}(\alpha) + \cdots + \beta_{NB} \varphi_{i,NB}(\alpha) + \varphi_{i,NB+1}(\alpha) - y_i, \tag{4}$$

where the $\varphi_{i,j}$'s are assumed to be continuously differentiable functions of α. Typically one will have $\varphi_{i,j} = \varphi_j(t_i)$ where t_i is a value of the independent variable in the data set. Examples of the formulation of problems as separable are given in Section C, page 12.

Where the symbols $\mathbf{x}$ and *nx* are used here they should be interpreted as denoting $\mathbf{c}$ and NC respectively in the nonseparable algorithms, and α and NA respectively in the separable algorithms. The notation $\|\cdot\|$ will denote the Euclidean vector norm throughout.

To denote subsets of the eight subroutines defined in Sections B.1 and B.2 we will use a "wild-card" notation. For example DNLAxx will denote any of the four subroutines of Section B.1.

B. Usage

B.1 Subroutines not Specialized for the Separable Problem: DNLAFB, DNLAFU, DNLAGB, DNLAGU

These subroutines are

DNLAFB Requires function values only. Applies bounds.

DNLAFU Requires function values only. No bounds.

DNLAGB Requires function and derivative values. Applies bounds.

DNLAGU Requires function and derivative values. No bounds.

For these subroutines the residual functions, r_i, are assumed to be continuously differentiable functions of the NC-vector **c**. Although the r_i's are not assumed to have any particular form, the form shown in Eq. (1) would be typical.

To conserve space the descriptions of these four subroutines are merged. Note that the argument BND() is only used by DNLAFB and DNLAGB, and the argument DCALCJ is only used by DNLAGU and DNLAGB.

B.1.a Program Prototype, Double Precision

INTEGER NDATA, NC, IV(LIV), LIV, LV

DOUBLE PRECISION COEF(≥NC), **BND**(2,≥NC), **V**(LV)

EXTERNAL DCALCR, DCALCJ

Assign values to NDATA, NC, COEF(), BND(), IV(), LIV, LV, and optionally to V().

CALL DNLAFU (NDATA, NC, COEF, DCALCR, IV, LIV, LV, V)

CALL DNLAGU (NDATA, NC, COEF, DCALCR, DCALCJ, IV, LIV, LV, V)

CALL DNLAFB (NDATA, NC, COEF, BND, DCALCR, IV, LIV, LV, V)

CALL DNLAGB (NDATA, NC, COEF, BND, DCALCR, DCALCJ, IV, LIV, LV, V)

Results are returned in COEF(), IV(), and V().

B.1.b Argument Definitions

NDATA [in] Number of observations (equations) in the problem. Require NDATA ≥ 1.

NC [in] Number of coefficients. Require NC ≥ 1. Generally one would have NC ≤ NDATA; however, NC > NDATA is permitted. In the latter case the solution will be nonunique, and if the computation is successful the return value of IV(1) is likely to be 7.

COEF() [inout] On entry, COEF(1:NC) must contain an initial estimate of the solution coefficients, denoted by **c** in Eqs. (1) and (2). On return COEF(1:NC) will contain the subroutine's best estimate of the solution coefficients.

BND() [in] Specifies bounds to be satisfied by the solution coefficients. The bounds are

$$\mathrm{BND}(1,j) \le \mathrm{COEF}(j) \le \mathrm{BND}(2,j),\ j = 1,\ldots,\ \mathrm{NC}$$

Require BND(1, j) ≤ BND(2,j) for j = 1, ..., NC. If a coefficient is to be unbounded in one or both directions, set the appropriate bound very small or very large, but not to such large magnitudes that the difference BND(2, j) − BND(1, j) would overflow. If the initial values for COEF( ) do not satisfy these bounds the coefficient values will be immediately altered so they are within the bounds.

DCALCR [in] The name of a subroutine provided by the user to compute the residual values, r_i, i = 1, ..., NDATA, using the current COEF() values. It must have an interface of the form

```
SUBROUTINE DCALCR(NDATA, NC, COEF, ICOUNT,
   RES)
INTEGER NDATA, NC, ICOUNT
DOUBLE PRECISION COEF(NC), RES(NDATA)
```

DCALCR must store the residual values in RES(i), i = 1, ..., NDATA. DCALCR must not change the values of NDATA, NC, or COEF().

In some applications DCALCR may need additional data, such as the t_i's or y_i's of Eq. (1), 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.

ICOUNT is a count of calls to DCALCR maintained by DNLAxx. DCALCR can reset ICOUNT to zero as a signal that it cannot evaluate the residual using the current COEF() values. This will cause DNLAxx to change the values in COEF() and try again.

DCALCJ [in] The name of a subroutine provided by the user to compute the Jacobian matrix of partial derivatives of the residual vector with respect to the coefficients using the current COEF() values. It must have an interface of the form

```
SUBROUTINE DCALCJ(NDATA, NC, COEF, ICOUNT,
   AJAC)
INTEGER NDATA, NC, ICOUNT
DOUBLE PRECISION COEF(NC), AJAC(NDATA, NC)
```

DCALCJ must store the value of the $\partial r_i/\partial c_j$ in AJAC(i, j), for i = 1, ..., NDATA, and j = 1, ..., NC. DCALCJ must not change the values of NDATA, NC, or COEF().

If DCALCJ needs additional data, that may have been input by the user's main program, such data can be made available to DCALCJ via named COMMON blocks.

ICOUNT is a count of calls to DCALCR maintained by DNLAxx. DCALCJ can reset ICOUNT to zero as a signal that it cannot evaluate the partial derivatives using the current COEF() values. This will cause DNLAxx to return to the calling program with IV(1) set to 65.

Generally the computation of the Jacobian matrix involves subexpressions that also appear in computing the residual vector. If efficiency is particularly important, the subroutines DCALCR and DCALCJ can be designed to avoid recalculation of such subexpressions. DCALCR can store values of common subexpressions into a COMMON block and copy ICOUNT into a COMMON variable, say KTAG, each time it is invoked. When DCALCJ is called it can compare KTAG from COMMON with ICOUNT from its argument list. When they are the same, which they will be most of the time, it means COEF() still has the same value it had when the subexpressions were computed, and thus DCALCJ can use these subexpression values without recomputing them.

For greatest efficiency DCALCR should save the two most recent evaluations, along with their ICOUNT values, say in KTAG1 and KTAG2 in a COMMON block. DCALCJ should then test ICOUNT against KTAG1 and KTAG2 and will usually find a match with one of them.

IV() [inout] An integer array of length LIV. Used as work space. Also used to input option selections and to output auxiliary results, including a beginning and ending status flag in IV(1). For simplest usage the user should set IV(1) = 0 before calling DNLAxx. This signals DNLAxx to set all options to default values by calling DIVSET. See Sections B.3 and B.4 for further information on setting options and interpreting output in IV().

LIV [in] The dimension of IV(). The minimum value allowed for LIV is for

DNLAFU, DNLAGU: 82 + NC,

and for

DNLAFB, DNLAGB: 82 + 4 × NC.

If LIV is set too small, DNLAxx will return with IV(1) = 15 and will store the minimal acceptable value for LIV in IV(*lastiv*) = IV(44), provided LIV ≥ 44.

LV [in] The dimension of V(). The minimum value allowed for LV is for

DNLAFU, DNLAGU: 105 + NC × (NDATA + 2 × NC + 17) + 2 × NDATA

and for

DNLAFB, DNLAGB: 105 + NC × (NDATA + 2 × NC + 21) + 2 × NDATA.

If LV is set too small, DNLAxx will return with IV(1) = 16 and will store the minimal acceptable value for LV in IV(*lastv*) = IV(45), provided LIV ≥ 45.

V() [inout] A floating point array of length LV. Used as work space. Also used to input option selections and to output auxiliary results. For simplest usage the user does not need to set any values in V() before calling DNLAxx. See Sections B.3 and B.4 for information on setting options and interpreting output in V().

B.2 Subroutines Specialized for the Separable Problem: DNLSFB, DNLSFU, DNLSGB, DNLSGU

These subroutines are

DNLSFB Requires function values only. Applies bounds to the nonlinear coefficients.

DNLSFU Requires function values only. No bounds.

DNLSGB Requires function and derivative values. Applies bounds to the nonlinear coefficients.

DNLSGU Requires function and derivative values. No bounds.

For these subroutines the residual functions, r_i, are assumed to be of the form shown in Eqs. (3) or (4), and the $\varphi_{i,j}$'s are assumed to be continuously differentiable functions of the NA-vector α.

To conserve space the descriptions of these four subroutines are merged. Note that the argument BND() is only used by DNLSFB and DNLSGB, and the argument DCALCB is only used by DNLSGU and DNLSGB.

B.2.a Program Prototype, Double Precision

INTEGER NDATA, NA, NB, IND(LIND,≥NA)**, LIND, IV**(LIV)**, LIV, LV**

DOUBLE PRECISION ALF(≥NA)**, BND**(2,≥NC)**, BET**(≥NB)**, YDATA**(≥NDATA)**, V**(LV)

EXTERNAL DCALCA, DCALCB

Assign values to NDATA, NA, NB, ALF(), BND(), YDATA(), IND(,), LIND, IV( ), LIV, LV, and optionally to V().

```
CALL DNLSFU (NDATA, NA, NB, ALF,
  BET, YDATA, DCALCA,
  IND, LIND, IV, LIV, LV, V)

CALL DNLSGU (NDATA, NA, NB, ALF,
  BET, YDATA, DCALCA, DCALCB,
  IND, LIND, IV, LIV, LV, V)

CALL DNLSFB (NDATA, NA, NB, ALF,
  BND, BET, YDATA, DCALCA,
            IND, LIND, IV, LIV, LV, V)

CALL DNLSGB (NDATA, NA, NB, ALF,
  BND, BET, YDATA, DCALCA,
  DCALCB, IND, LIND, IV, LIV, LV, V)
```

Results are returned in ALF(), BET(), IV(), and V().

B.2.b Argument Definitions

NDATA [in] Number of observations (equations) in the problem. Require NDATA ≥ 1.

NA [in] Number of nonlinear coefficients. Require NA ≥ 1.

NB [in] Number of linear coefficients. Require NB ≥ 0. Generally one would have NA + NB $\leq$ NDATA; however, NA + NB > NDATA is permitted. In the latter case the solution will be nonunique, and if the computation is successful the return value of IV(1) is likely to be 7.

ALF() [inout] On entry, ALF(1:NA) must contain an initial estimate of the nonlinear coefficients, denoted by α in Eqs. (3) and (4). On return ALF(1:NA) will contain the subroutine's best estimate of the α vector.

BND() [in] Specifies bounds to be satisfied by the nonlinear coefficients. The bounds are

$$\text{BND}(1,j) \leq \text{ALF}(j) \leq \text{BND}(2,j),\ j = 1,\ ...,\ \text{NA}$$

Require BND$(1,j) \leq$ BND$(2,j)$ for $j = 1$, ..., NA. If a coefficient is to be unbounded in one or both directions, set the appropriate bound very small or very large, but not to such large magnitudes that the difference BND$(2,j) -$ BND$(1,j)$ would overflow. If the initial values for ALF() do not satisfy these bounds the coefficient values will be immediately altered so they are within the bounds.

BET() [out] On return BET(1:NB) will contain the values of the linear coefficients, denoted by β in Eqs. (3) and (4).

YDATA() [in] On entry YDATA(1:NDATA) must contain the data values denoted by y_i in Eqs. (3) and (4).

DCALCA [in] The name of a subroutine provided by the user to compute values of the functions $\varphi_{i,j}$ of Eqs. (3) or (4), using the current ALF() values. It must have an interface of the form

```
SUBROUTINE DCALCA(NDATA, NA, NB, ALF,
   ICOUNT, PHI)
INTEGER NDATA, NA, NB, ICOUNT
DOUBLE PRECISION ALF(NA), PHI(NDATA, NB or
   NB+1)
```

DCALCA must store the value of $\varphi_{i,j}$ in PHI(i,j), for $i = 1$, ..., NDATA, and $j = 1$, ... NB or NB+1. The array PHI() must have NB columns if the model is Eq. (3), and NB+1 columns if it is Eq. (4).

DCALCA must not change the values of NDATA, NA, NB, or ALF().

If DCALCA needs additional data, which may have been input by the user's main program, such data can be made available to DCALCA via named COMMON blocks.

ICOUNT is a count of calls to DCALCA maintained by DNLSxx. DCALCA can reset ICOUNT to zero as a signal that it cannot compute PHI() using the current ALF() values. This will cause DNLSxx to change the values in ALF() and try again.

DCALCB [in] The name of a subroutine provided by the user to compute partial derivatives of the functions $\varphi_{i,j}$ with respect to the components of the coefficient vector α. It must have an interface of the form

```
SUBROUTINE DCALCB(NDATA, NA, NB, ALF,
   ICOUNT, DER)
INTEGER NDATA, NA, NB, ICOUNT
DOUBLE PRECISION ALF(NA), DER(NDATA, nzero)
```

where *nzero* is the number of nonzeros in the array IND(,). See the specification of IND(,) below.

Let the nonzeros in IND(,) be indexed from 1 to *nzero* as they occur in traversing down the first column, then the second column, etc. If the m^{th} nonzero in this ordering is at IND(j,k), then DCALCB must store the value of $\partial\varphi_{i,j}/\partial\alpha_k$ in DER(i,m) for $i = 1$, ..., NDATA, and $m = 1$, ..., *nzero*.

Equivalently we may describe DER() by saying that row i of DER() must contain values of $\partial\varphi_{i,j}/\partial\alpha_k$ for all index pairs (j,k) for which α_k appears in the formula defining $\varphi_{\ell,j}$ for at least some ℓ, and these values must be ordered across row i of DER() with major sort on k and minor sort on j.

DCALCB must not change the values of NDATA, NA, NB, or ALF().

If DCALCB needs additional data, which may have been input by the user's main program, such data can be made available to DCALCB via named COMMON blocks.

ICOUNT is a count of calls to DCALCA maintained by DNLSxx. DCALCB can reset ICOUNT to zero as a signal that it cannot evaluate the partial derivatives using the current ALF() values. This will cause DNLSxx to return to the calling program with IV(1) set to 65.

Generally the computation of the derivatives involves subexpressions that also appear in computing the $\varphi_{i,j}$'s. If efficiency is important, the subroutines DCALCA and DCALCB can be designed to avoid recalculation of such subexpressions. DCALCA can store values of common subexpressions into a COMMON block and copy ICOUNT into a COMMON variable, say KTAG, each time it is invoked. When

DCALCB is called it can compare KTAG from COMMON with ICOUNT from its argument list. When they are the same, which they will be most of the time, it means ALF() still has the same value it had when the subexpressions were computed, and thus DCALCB can use these subexpression values without recomputing them.

For greatest efficiency DCALCA should save the two most recent evaluations, along with their ICOUNT values, say in KTAG1 and KTAG2 in a COMMON block. DCALCB should then test ICOUNT against KTAG1 and KTAG2 and will usually find a match with one of them.

IND(,) [in] An integer array containing an (NB+1)×NA matrix of zeros and ones. The user must set IND$(j,k) = 1$ if the coefficient α_k appears in the formula defining $\varphi_{i,j}$, for some i, and set IND$(j,k) = 0$ otherwise. If the model of Eq. (3) is used, all elements of IND(NB+1, 1:NA) must be set to zero. The condition of IND(NB+1, 1:NA) being all zero signals DNLSxx that Eq. (3) rather than Eq. (4) is being used.

If any column of IND(,) were entirely zero it would mean the corresponding component of α would not affect the model function. This is regarded as an error condition and will cause a return with IV(1) = 66.

LIND [in] The first dimensioning parameter for IND(,). Require LIND ≥ NB + 1.

IV() [inout] An integer array of length LIV. Used as work space. Also used to input option selections and to output auxiliary results, including a beginning and ending status flag in IV(1). For simplest usage the user should set IV(1) = 0 before calling DNLSxx. This signals DNLSxx to set all options to default values by calling DIVSET. See Sections B.3 and B.4 for further information on setting options and interpreting output in IV().

LIV [in] The dimension of IV(). Let *nzero* denote the number of nonzeros in IND(,).

Program	Minimum value allowed for LIV
DNLSFU	$122 + 2 \times nzero + 4 \times \text{NA} + 2 \times \text{NB} + \max(\text{NB}+1, 6 \times \text{NA})$
DNLSGU	$115 + \text{NA} + \text{NB} + 2 \times nzero$
DNLSFB	$122 + 2 \times nzero + 7 \times \text{NA} + 2 \times \text{NB} + \max(\text{NB}+1, 6 \times \text{NA})$
DNLSGB	$115 + 4 \times \text{NA} + \text{NB} + 2 \times nzero$

If LIV is set too small, DNLSxx will return with IV(1) = 15 and will store the minimal acceptable value for LIV in IV(*lastiv*) = IV(44), provided LIV ≥ 44.

LV [in] The dimension of V(). Define:

nzero = the number of nonzeros in IND(,),

bterm = (NB × (NB+3))/2, *nc* = NA + NB, and

jlen = NA × NDATA if neither the covariance matrix nor regression diagnostics are requested, otherwise *jlen* = (NB + NA) × (NDATA + NB + NA + 1).

Program	Minimum value allowed for LV
DNLSFU	$105 + 2 \times \text{NDATA} \times (\text{NB}+3) + jlen + bterm + \text{NA} \times (2 \times \text{NA} + 18)$
DNLSGU	$105 + \text{NDATA} \times (\text{NB} + nzero + 3) + jlen + bterm + \text{NA} \times (2 \times \text{NA} + 17)$
DNLSFB	$105 + \text{NDATA} \times (2 \times \text{NB} + 6 + \text{NA}) + bterm + \text{NA} \times (2 \times \text{NA} + 22)$
DNLSGB	$105 + \text{NDATA} \times (\text{NA} + \text{NB} + nzero + 3) + bterm + \text{NA} \times (2 \times \text{NA} + 21)$

If the problem is of the form of Eq. (3) rather than Eq. (4), LV can be less than indicated above by 4 × NDATA for DNLSFU or DNLSFB, and by NDATA for DNLSGU or DNLSGB.

If LV is set too small, DNLSxx will return with IV(1) = 16 and will store the minimal acceptable value for LV in IV(*lastv*) = IV(45), provided LIV ≥ 45.

V() [inout] A floating point array of length LV. Used as work space. Also used to input option selections and to output auxiliary results. For simplest usage the user does not need to set any values in V() before calling DNLSxx. See Sections B.3 and B.4 for information on setting options and interpreting output in V().

B.3 Setting Options and Using Subroutine DIVSET

Options are selected by storing values into IV() and V(). To set options to nondefault values you should first call DIVSET which will store default values into IV() and V(), then individually reset particular elements of IV() and V() to desired nondefault values, and then call the desired DNLxxx subroutine.

We describe here the call to DIVSET. The contents of IV() and V() are described in Section B.4.

B.3.a Program Prototype, Double Precision

INTEGER MODE, IV(LIV), LIV, LV

DOUBLE PRECISION V(LV)

Set MODE = 1, and assign values to LIV and LV.

CALL DIVSET (MODE, IV, LIV, LV, V)

Results are returned in IV() and V().

B.3.b Argument Definitions

MODE [in] Always set MODE = 1.

IV() [out] Integer array into which DIVSET will store default values. On return will have IV(1) = 12.

LIV [in] Dimension of IV(). DIVSET requires LIV $\geq$ 82, but a larger value will be needed for DNLxxx.

LV [in] Dimension of V(). DIVSET requires LV $\geq$ 98, but a larger value will be needed for DNLxxx.

V() [out] Floating point array into which DIVSET will store default values.

B.4 The Contents of IV() and V()

The following Sections, B.4.a to B.4.j, give the interpretation of the elements of IV() and V() that are most likely to be of interest to the user of this package.

The values that will be assigned by DIVSET are indicated by "Default = ...". See Section D for discussion of objects such as the **d** vector, the S matrix, the Gauss-Newton and augmented models, etc.

The name *machep* will be used in the following descriptions to denote the unit round-off of the host arithmetic. It is obtained in the code by calling D1MACH(4) (or R1MACH(4) for single precision); see Chapter 19.1.

Internally this package uses symbolic names for certain fixed index values used in IV() or V(). These associations will be indicated in the following descriptions by the notation: IV(*covprt*) $\equiv$ IV(14). If the user wishes to use such an association in his/her program the Fortran 77 syntax would be

INTEGER *covprt*

PARAMETER(*covprt* = 14)

The contents of IV() and V() will be described in functional groupings in Sections B.4.a through B.4.j.

B.4.a Primary entry mode and termination status flag: IV(1).

B.4.b Primary output objects: Objective function value, Gradient vector: V(f), IV(g).

B.4.c Print options: IV(*prunit*), IV(*covprt*), IV(*outlev*), IV(*parprt*), IV(*solprt*), IV(*statpr*), IV(*x0prt*).

B.4.d Convergence tolerances: V(*afctol*), V(*lmaxs*), V(*rfctol*), V(*sctol*), V(*xctol*), V(*xftol*).

B.4.e Operation count limits: IV(*mxfcal*), IV(*mxiter*).

B.4.f The covariance matrix and regression diagnostics: IV(*rdreq*), IV(*covreq*), IV(*covmat*), IV(*regd*).

B.4.g Parameters used in approximating derivatives by differences: V(*delta0*), V(*dltfdc*), V(*dltfdj*).

B.4.h The scaling vector **d**: IV(*dtype*), V(*dfac*), V(*dinit*), V(*d0init*), V(*dtinit*), IV(d), IV(*jtol*).

B.4.i The S matrix: IV(*inits*), IV(s)

B.4.j Additional output quantities: IV(*lastiv*), IV(*lastv*), IV(*nfcall*), IV(*nfcov*), IV(*ngcall*), IV(*ngcov*), IV(*niter*), V(*dgnorm*), V(*dstnrm*), V(*f0*), V(*nreduc*), V(*preduc*), V(*rcond*), V(*reldx*), V(*stppar*).

The following table summarizes the indices used.

Indices used in IV()			Indices used in V()		
Name	**Value**	**Ref.**	**Name**	**Value**	**Ref.**
	1	B.4.a	*afctol*	31	B.4.d
covmat	26	B.4.f	*d0init*	40	B.4.h
covprt	14	B.4.c	*delta0*	44	B.4.g
covreq	14	B.4.c	*dfac*	44	B.4.g
d	27	B.4.h	*dgnorm*	1	B.4.j
dtype	16	B.4.h	*dinit*	38	B.4.h
g	28	B.4.b	*dltfdc*	42	B.4.g
inits	25	B.4.i	*dltfdj*	43	B.4.g
jtol	59	B.4.h	*dstnrm*	2	B.4.j
lastiv	44	B.4.j	*dtinit*	39	B.4.h
lastv	45	B.4.j	*f*	10	B.4.b
mxfcal	17	B.4.e	*f0*	13	B.4.j
mxiter	18	B.4.e	*lmaxs*	36	B.4.d
nfcall	6	B.4.j	*nreduc*	6	B.4.j
nfcov	52	B.4.j	*preduc*	7	B.4.j
ngcall	30	B.4.j	*rcond*	53	B.4.j
ngcov	53	B.4.j	*reldx*	17	B.4.j
niter	31	B.4.j	*rfctol*	32	B.4.d
outlev	19	B.4.c	*sctol*	37	B.4.d
parprt	20	B.4.c	*stppar*	5	B.4.j
prunit	21	B.4.c	*xctol*	33	B.4.d
rdreq	57	B.4.f	*xftol*	34	B.4.d
regd	67	B.4.f			
s	62	B.4.i			
solprt	22	B.4.c			
statpr	23	B.4.c			
x0prt	24	B.4.c			

B.4.a Primary entry mode and termination status flag: IV(1).

On entry to DNLxxx, IV(1) specifies the entry mode. Require $0 \leq$ IV(1) ≤ 14.

Zero means the user is not setting any options and DNLxxx is to call DIVSET to set all options to default values and proceed to execute the problem-solving algorithm.

A value from 1 to 11 means DNLxxx has returned to the user with this value in IV(1), and the user's program has changed some tolerance(s) and wishes to resume computation without starting over from scratch.

12 means all options have already been set in IV() and V() and DNLxxx is not to call DIVSET. Typically the

user will have called DIVSET to set default values into IV() and V() and then may have altered some from the default settings. DIVSET sets IV(1) = 12.

13 means the user has set all options, as described above for 12, but also wishes to set one or more of the vectors **d**, **d0**, or **dtol** or the matrix, S. DNLxxx will set IV(d), IV($jtol$), and IV(s) that depend on the problem size, and then return, setting IV(1) = 14. The user can then set one or more of

d in V() starting at V(IV(d)),

d0 in V() starting at V(IV($jtol$)+nx), where nx denotes NC or NA,

dtol in V() starting at V(IV($jtol$)), and

S stored in V() by rows of the lower triangle, starting at V(IV(s)),

and then reenter DNLxxx with IV(1) = 14. See IV($dtype$) and IV($inits$) for additional information.

14 means the process described above for IV(1) = 13 has been done. DNLxxx will assume all options have been set and will proceed to execute the problem-solving algorithm.

On return, IV(1) indicates the reason for the return. Values 3–6 indicate successful termination. Values 7–14 permit continuation after changing some input values in IV() or V(). Values exceeding 14 do not permit continuation.

3 Coefficient convergence. The scaled relative difference (See Eq. (5), page 9) between the current **c** or α and a locally optimal parameter vector is very likely at most V($xctol$).

4 Relative function convergence. The relative difference between the current objective function value and its locally optimal value is very likely at most V($rfctol$).

5 Both coefficient and relative function convergence (*i.e.*, the conditions for IV(1) = 3 and IV(1) = 4 both hold).

6 Absolute function convergence. The current objective function value is at most V($afctol$) in absolute value.

7 Singular convergence. The Hessian near the current iterate appears to be singular or nearly so, and a step of length at most V($lmaxs$) is unlikely to yield a relative function decrease of more than V($sctol$). This could be a successful termination when NC > NDATA or NA + NB > NDATA. Otherwise check for errors in the problem formulation, or consider reducing the number of coefficients to be determined.

8 False convergence. The iterates appear to be converging to a noncritical point. This may mean that the convergence tolerances (V($afctol$), V($rfctol$), V($xctol$)) are too small for the accuracy to which the function and gradient are being computed, that there is an error in computing the gradient, or that the function or gradient is discontinuous near the current **c** or α.

9 Function evaluation count limit reached without other convergence. See IV($mxfcal$).

10 Iteration count limit reached without other convergence. See IV($mxiter$).

11 External interrupt via subprogram STOPX. This is not activated in the MATH77 version.

14 DNLxxx was entered with IV(1) = 13.

15 LIV is too small. An acceptable value is given in IV($lastiv$) if LIV $\geq$ $lastiv$ (=44).

16 LV is too small. An acceptable value is given in IV($lastv$) if LIV $\geq$ $lastv$ (=45).

17 Restart attempted with NDATA, NC, NA, or NB changed. This is not permitted.

18 Bad initialization of the scaling vector **d**. Check IV($dtype$), IV($dinit$), and V(IV(d):IV(d)+nx−1) where nx denotes NC for DNLAxx and NA for DNLSxx.

19...44 V(IV(1)−18) is out of range.

63 **r**(**c**) or **r**(α, β) cannot be computed at the initial **c** or α.

64 Bad parameters on an internal call. Should not happen.

65 The derivatives could not be computed at the current **c** or α (see DCALCJ or DCALCB).

66 Some column of IND(,) is all zero, or some array dimensions are inconsistent.

67 Call to DIVSET with argument MODE not set to 1.

68, 69 Internal errors. Should not happen.

70 Inconsistent bounds. Require BND(1, j) ≤ BND(2, j) for j = 1, ..., nx, where nx = NC or NA.

80 IV(1) exceeds 14 on entry.

81 NDATA, NC, NA, or NB out of range. Require NDATA > 0, NC > 0, NA > 0, and NB ≥ 0.

87...(86 + *nx*) Bad initialization of **dtol**. Check V($dtinit$) and V(IV($jtol$):IV($jtol$) + nx − 1), where nx denotes NC or NA.

B.4.b Primary output objects: Objective function value, Gradient vector: V(f), IV(g).

V(f) ≡ V(10) is the current value of the objective function ψ defined by Eq. (2), *i.e.*, half the sum of squares of residuals.

IV(g) ≡ IV(28) is the starting subscript in V() of the current gradient vector $\mathbf{g} = J^t\mathbf{r}$.

B.4.c Print options: IV(*prunit*), IV(*covprt*), IV(*outlev*), IV(*parprt*), IV(*solprt*), IV(*statpr*), IV(*x0prt*).

IV(*prunit*) ≡ IV(21) = 0 or the output unit number on which all printing is done. The default is the standard output unit number obtained from I1MACH(2) (See Chapter 19.1.), which will be 6 on most systems. 0 means suppress all printing.

IV(*covprt*) ≡ IV(14) Selects printing of the covariance matrix and/or the regression diagnostics when the solution process is successfully completed, *i.e.*, on a return with IV(1) = 3, 4, 5, or 6. Relevant only for DNLxxU for the reasons given in the description of IV(*rdreq*). Allowed values are 0, 1, 2, or 3 and the default is 0.

- 0 means neither is to be printed.
- 1 means print the covariance matrix.
- 2 means print the regression diagnostics.
- 3 means print both.

Selection of printing also causes the selected object to be computed even if it was not selected by IV(*rdreq*). In such a case the method of computing the covariance matrix is that of IV(*covreq*) = 1.

IV(*outlev*) ≡ IV(19) controls printing of iteration summary lines. Default = 0.

IV(*outlev*) = 0 means do not print any summary lines. Otherwise, print a summary line after each abs(IV(*outlev*)) iterations.

If IV(*outlev*) is positive, then summary lines of length 117 (plus carriage control) are printed.

If IV(*outlev*) is negative, lines of maximum length 79 (or 55 if IV(*covprt*) = 0) are printed, including only the first 6 items listed below (through V(*reldx*)).

The items printed will be:

The iteration and function evaluation counts: IV(*niter*) and IV(*nfcall*).

Current value of ψ: V(f).

Relative change in ψ achieved by the latest step, *i.e.*, RELDF = (V(*f0*) − V(f))/V(*f0*),

The relative objective function reduction predicted for the step just taken, *i.e.*, PRELDF = V(*preduc*)/V(*f0*),

The scaled relative change in **c** or α: V(*reldx*),

The model used in the current iteration (G = Gauss-Newton, S = augmented),

The Marquardt parameter V(*stppar*) used in computing the last step,

The sizing factor used in updating S (see [1]),

The 2-norm of the scale vector **d** times the step just taken (see [1]), and

NPRELDF = V(*nreduc*)/V(*f0*).

If NPRELDF is positive, then it is the relative function reduction predicted for a Newton step (one with *stppar* = 0).

If NPRELDF is zero, either the gradient vanishes (as does PRELDF) or else the augmented model is being used and its Hessian is indefinite (with PRELDF positive).

If NPRELDF is negative, then it is the negative of the relative function reduction predicted for a step computed with step bound V(*lmaxs*) for use in testing for singular convergence.

IV(*parprt*) ≡ IV(20) = 0 or 1. Default = 0. 1 means print any nondefault V() values on a fresh start or any changed V() values on a restart. 0 means skip this printing.

IV(*solprt*) ≡ IV(22) = 0 or 1. Default = 0. 1 means print the solution, **c** for DNLAxx or α and β for DNLSxx, as well as the gradient vector, **g**, and final scaling vector, **d**. 0 means skip this printing.

IV(*statpr*) ≡ IV(23) = 0 or 1. Default = 0. 1 means print summary statistics upon returning. 0 means skip this printing. The items printed are:

The final value of ψ:V(f).

V(*reldx*).

FUNC. EVALS: Number of calls to DCALCR or DCALCA for function evaluations.

GRAD. EVALS: In DNLxGx this is the number of calls to DCALCJ or DCALCB. In DNLxFX this is the number of calls to DCALCR or DCALCA for computation of finite-difference approximations to the gradient.

The relative function reductions predicted for the last step taken and for a Newton step, or perhaps a step bounded by V(*lmaxs*). See the descriptions of PRELDF and NPRELDF under IV(*outlev*) above.

The number of calls to DCALCR and DCALCJ or DCALCA and DCALCB used in trying to compute the covariance matrix, if an attempt was made to compute it.

IV(*x0prt*) ≡ IV(24) = 0 or 1. Default = 0. 1 means print the initial **c** or α and scale vector **d** (on a fresh start only). 0 means skip this printing.

B.4.d Convergence tolerances: V(*afctol*), V(*lmaxs*), V(*rfctol*), V(*sctol*), V(*xctol*), V(*xftol*).

V(*afctol*) ≡ V(31) is the absolute function convergence tolerance. If DNLxxx finds a point where ψ is less than V(*afctol*), and if DNLxxx does not return with IV(1) = 3, 4, or 5, then it returns with 8(?).

V(*lmaxs*) ≡ V(36) Default = 1.0. Used with V(*sctol*), see below.

V(*rfctol*) ≡ V(32) is the relative function convergence tolerance. If the current model predicts a maximum possible function reduction (see V(*nreduc*)) of at most V(*rfctol*) × ψ_0 at the start of the current iteration, where ψ_0 is the current function value, and if the last step attempted achieved no more than twice the predicted function decrease, then DNLxxx returns with IV(1) = 4 (or 5). Default = $\max(10^{-10}, machep^{2/3})$.

V(*sctol*) ≡ V(37) is the singular convergence tolerance. Default = $\max(10^{-10},\ machep^{2/3})$. Singular convergence occurs if the tests for returns with IV(1) = 3, 4, 5, or 6 are not satisfied but it appears that no step with scaled step length less than V(*lmaxs*) can make a change of more than V(*sctol*) × ψ in the value of ψ.

V(*xctol*) ≡ V(33) is the coefficient convergence tolerance. Default = $\sqrt{machep}$. Coefficient convergence occurs if the algorithm thinks the scaled relative distance from the current **c** or α to the solution is at most V(*xctol*), where the scaled relative distance between two vectors **x** and **y** is computed as

$$\frac{\max_j \{d_j|x_j - y_j|\}}{\max_j \{d_j\,(|x_j| + |y_j|)\}} \tag{5}$$

This distance function is computed by subprogram DRLDST. One could replace this subprogram to use a different distance function. DRLDST has an interface of the form

```
DOUBLE PRECISION FUNCTION DRLDST(NX,D,X,Y)
INTEGER NX
DOUBLE PRECISION D(NX), X(NX), Y(NX)
```

V(*xftol*) ≡ V(34) is the false convergence tolerance. Default = 100 × *machep*. False convergence occurs if the tests for returns with IV(1) = 3, 4, 5, 6, or 7 are not satisfied and a step of scaled relative length (measured using Eq. (5)) at most V(*xftol*) is tried but not accepted. See remarks for IV(1) = 8.

B.4.e Operation count limits: IV(*mxfcal*), IV(*mxiter*).

IV(*mxfcal*) ≡ IV(17) gives the maximum number of function evaluations (calls to DCALCR or DCALCA, excluding those used to compute the covariance matrix) allowed. If this number does not suffice, DNLxxx returns with IV(1) = 9. Default = 200.

IV(*mxiter*) ≡ IV(18) gives the maximum number of iterations allowed. It also indirectly limits the number of gradient evaluations (calls to DCALCJ or DCALCB, excluding those used to compute the covariance matrix) to IV(*mxiter*) + 1. If IV(*mxiter*) iterations do not suffice, then DNLxxx returns with IV(1) = 10. Default = 150.

B.4.f The covariance matrix and regression diagnostics: IV(*rdreq*), IV(*covreq*), IV(*covmat*), IV(*regd*).

IV(*rdreq*) requests computation of the covariance matrix and/or regression diagnostics. IV(*covreq*) specifies how the covariance matrix is to be computed. The storage locations of these objects are indicated by IV(*covmat*) and IV(*regd*). Printing of these objects is selected by IV(*covprt*) described in Section B.4.c.

IV(*rdreq*) ≡ IV(57) Selects computation of the covariance matrix and/or the regression diagnostics described in Section D.

IV(*rdreq*), IV(*covprt*) and IV(*covreq*) are relevant only with DNLxxU, and have no effect when using DNLxxB, since the concepts of a covariance matrix and regression diagnostics are less well-defined when variables are bounded. See, *e.g.*, [2, pp 180–183] for a discussion of this issue.

Possible values are 0, 1, 2, and 3 and the default is 3.

- 0 means do not compute these.
- 1 means compute just the covariance matrix.
- 2 means compute just the regression diagnostics.
- 3 means compute both the covariance matrix and the regression diagnostics.

The covariance matrix is a symmetric matrix of order NC for the DNLAxU, and of order NA + NB for DNLSxU. In the latter case the first NA indices in the covariance matrix are associated with the α vector while the last NB indices are associated with the β vector. See IV(*covmat*) and IV(*regd*) for specification of the storage methods for these objects.

IV(*covreq*) ≡ IV(15) Specifies how the covariance matrix is to be computed if its computation is requested by IV(*rdreq*) or by IV(*covprt*). Relevant only for DNLxxU for the reasons given in the description of IV(*rdreq*). For DNLxGU this should be set to 1, 2, or 3 and the default is 1. For DNLxFU it should be set to −1, −2, or −3, and the default is −1. If it is set positive for DNLxFU it will be reset to negative.

Let K = |IV(*covreq*)| and let

$$\text{VARFAC} = 2\psi/\text{DOF},$$

where ψ is defined by Eq. (2), and DOF = NDATA − NC for DNLAxU and NDATA − NA − NB for DNLSxU.

If K = 1 or 2, then a finite-difference Hessian approximation H is obtained. If H is positive definite (or, for K = 3, if the Jacobian matrix J is of full rank), one of the following is computed:

K = 1 ⇒ VARFAC × $H^{-1}J^tJH^{-1}$.

K = 2 ⇒ VARFAC × H^{-1}.

K = 3 ⇒ VARFAC × $(J^tJ)^{-1}$.

If IV(*covreq*) > 0, H will be computed using first differences of derivative values with step sizes determined using V(*delta0*). This requires a user-provided DCALCJ or DCALCB subroutine.

If IV(*covreq*) < 0, H will be computed using second differences of function values with step sizes determined using V(*dltfdc*).

(IV(*covreq*) = 0 will act the same as IV(*covreq*) = 1, but this is not recommended.)

IV(*covmat*) ≡ IV(26) tells whether a covariance matrix was computed. If (IV(*covmat*) is positive, the lower triangle of the covariance matrix is stored compactly by rows in V() starting at V(IV(*covmat*)). If IV(*covmat*) = 0, no attempt was made to compute the covariance. If IV(*covmat*) = −1, the finite-difference Hessian was indefinite. If IV(*covmat*) = −2, a successful finite-differencing step could not be found for some component of **c** or α (*i.e.*, DCALCR or DCALCA set ICOUNT to 0 for each of two trial steps).

IV(*regd*) ≡ IV(67) If nonzero, this is the starting subscript in V() of the NDATA regression diagnostics. 0 means the regression diagnostics were not computed (either not requested or the computation failed.)

B.4.g Parameters used in approximating derivatives by differences: V(*delta0*), V(*dltfdc*), V(*dltfdj*).

V(*delta0*) ≡ V(44) Used in choosing the step size for computing the Hessian matrix by differencing gradient values. Used when IV(*inits*) = 3 or IV(*covreq*) = 1 or 2. For component j, the step size

$$\text{V}(\textit{delta0}) \times \max(|x_j|, 1/d_j) \times \text{ sign}(x_j)$$

is used. If this step results in DCALCR or DCALCA setting ICOUNT to 0, then −0.5 times this step is also tried. Default = $\sqrt{machep}$.

V(*dltfdc*) ≡ V(42) Used in choosing the step size for computing the Hessian matrix by second differences of function values. Used when IV(*inits*) = 3 or IV(*covreq*) = −1 or −2. For differences involving x_j, the step size first tried is

$$\text{V}(\textit{dltfdc}) \times \max(|x_j|, 1/d_j)$$

If this step is too big the first time it is tried, *i.e.*, if DCALCR or DCALCA sets ICOUNT to 0, then −0.5 times this step is also tried. Default = $machep^{1/3}$.

V(*dltfdj*) ≡ V(43) Used in choosing the step size for computing the Jacobian matrix by differencing function values. Default = $\sqrt{machep}$. The first step tried when differencing in the x_j coordinate will be

$$\text{V}(\textit{dltfdj}) \times \max(|x_j|, 1/d_j)$$

If this step causes DCALCR or DCALCA to set ICOUNT = 0, smaller steps will be tried.

B.4.h The scaling vector d: IV(*dtype*), V(*dfac*), V(*dinit*), V(*d0init*), V(*dtinit*), IV(d), IV(*jtol*).

IV(*dtype*) ≡ IV(16) = 0, 1, or 2. Default = 1. Specifies if and when the scaling vector **d**, discussed in Section D, is to be updated. The vector **d** is of dimension *nx* where *nx* = NC for DNLAxx and NA for DNLSxx.

- 0 means **d** is not to be updated. In this case the user must either store a positive value in V(*dinit*), which will be assigned as the value of all components of **d**, or must set V(*dinit*) to a negative value, say −1.0, and store the complete **d** vector in V() beginning at V(IV(d)). To give the complete **d** vector the user must use the process described above for IV(1) = 13 in order to have the package set IV(d).
- 1 means **d** is to be updated at every iteration.
- 2 means **d** is to be updated only at the first iteration and left at that setting throughout the rest of the solution procedure.

The updating procedure uses the number V(*dfac*) and two *nx*-dimensional vectors **d0** and **dtol**. This updating is done in subroutine DD7UPD. The updating algorithm is

$$d_j = \max(\text{V}(\textit{dfac}) \times d_j, jcnorm_j)$$
if $d_j < dtol_j$ then $d_j = d0_j$

Here $jcnorm_j$ denotes the computed Euclidean norm of the j^{th} column of the current Jacobian matrix.

If V(*dinit*) ≥ 0, all components of **d** will initially be set to this value. Otherwise the user must supply the

complete **d** vector, beginning in V(IV(d)), using the process described for IV(1) = 13.

If V(*d0init*) > 0, all components of **d0** will be set to this value. Otherwise the user must supply the complete **d0** vector, beginning in V(IV(*jtol*) + nx), using the process described for IV(1) = 13.

If V(*dtinit*) > 0, all components of **dtol** will be set to this value. Otherwise the user must supply the complete **dtol** vector, beginning in V(IV(*jtol*)), using the process described for IV(1) = 13.

V(*dfac*) ≡ V(41) Default = 0.6. See IV(*dtype*).

V(*dinit*) ≡ V(38) Default = 0. See IV(*dtype*).

V(*d0init*) ≡ V(40) Default = 1.0. See IV(*dtype*)

V(*dtinit*) ≡ V(39) Default = 10^{-6}. See IV(*dtype*).

IV(d) ≡ IV(27) is the starting subscript in V() of the current scale vector **d**. See IV(*dtype*).

IV(*jtol*) ≡ IV(59) is the starting subscript in V() of the vector **dtol**, and IV(*jtol*) + nx is the starting subscript in V() of the vector **d0**. See IV(*dtype*).

B.4.i The S matrix: IV(*inits*), IV(s)

IV(*inits*) ≡ IV(25) = 0, 1, 2, 3, or 4. Default = 0. Chooses how the S matrix discussed in Section D should be initialized.

- 0 means initialize S to 0 and start with the Gauss-Newton model.
- 1 means the user is providing the initial S and start with the Gauss-Newton model.
- 2 means the user is providing the initial S and start with the augmented model.
- 3 or 4 mean the package is to compute a finite difference approximation to the initial S and start with the augmented model. 3 means the package will compute second differences of function values using the parameter V(*dltfdc*).
- 4 which can only be used with DNLxGx, means the package will compute first differences of gradient values using the parameter V(*delta0*).

In cases 1 or 2 the caller must store the lower triangle of the initial S, compactly by rows, in V() starting at V(IV(s)). To do this the user must use the procedure described above for IV(1) = 13.

IV(s) ≡ IV(62) is the starting subscript in V() of the S matrix. It is stored by rows of the lower triangle.

B.4.j Additional output quantities: IV(*lastiv*), IV(*lastv*), IV(*nfcall*), IV(*nfcov*), IV(*ngcall*), IV(*ngcov*), IV(*niter*), V(*dgnorm*), V(*dstnrm*), V(*f0*), V(*nreduc*), V(*preduc*), V(*rcond*), V(*reldx*), V(*stppar*).

IV(*lastiv*) ≡ IV(44) is the minimal acceptable value for LIV. Set only on a return with IV(1) = 15.

IV(*lastv*) ≡ IV(45) is the minimal acceptable value for LV. Set only on a return with IV(1) = 16.

IV(*nfcall*) ≡ IV(6) is the number of calls made to DCALCR or DCALCA, *i.e.*, function evaluations, including those used in computing the covariance.

IV(*nfcov*) ≡ IV(52) is the number of calls made to DCALCR or DCALCA when computing the covariance matrix.

IV(*ngcall*) ≡ IV(30) is the number of gradient evaluations, *i.e.*, calls to DCALCJ or DCALCB, including those used for computing the covariance.

IV(*ngcov*) ≡ IV(53) is the number of calls made to DCALCJ or DCALCB when computing the covariance matrix.

IV(*niter*) ≡ IV(31) is the number of iterations performed.

V(*dgnorm*) ≡ V(1) = $\|D^{-1}\mathbf{g}\|$ where **g** is the gradient vector.

V(*dstnrm*) ≡ V(2) = $\|D \times (\mathbf{x} - \mathbf{x}^{prev})\|$, *i.e.*, the scaled length of the most recent step. Here **x** denotes **c** or α.

V(*f0*) ≡ V(13) is the value of ψ at the end of the previous iteration.

V(*nreduc*) ≡ V(6) if positive, or zero with V(*stppar*) = 0, is the maximum objective function reduction possible according to the current model. V(*nreduc*) = 0 with V(*stppar*) > 0 means H is not positive definite.

If V(*nreduc*) < 0, then −V(*nreduc*) / V(*f0*) is the quantity with which V(*sctol*) is compared in the singular convergence test.

V(*preduc*) ≡ V(7) is the function reduction predicted for the last step taken, or attempted. V(*preduc*)/ max(V(f), V(*f0*)) is used in testing for relative function convergence.

V(*rcond*) ≡ V(53) If |IV(*covreq*)| = 1 or 2 this is an approximate reciprocal condition number of the finite-difference Hessian approximation. If |IV(*covreq*)| = 3 this is an approximate reciprocal condition number of the current Jacobian matrix.

V(*reldx*) ≡ V(17) is the scaled relative change in **c** or α due to the last step taken or attempted, computed using Eq. (5).

V(*stppar*) ≡ V(5) If nonnegative, this is the Levenberg-Marquardt parameter, λ, of Eq. (19) Thus, a zero value indicates an undamped Newton step. A negative value indicates a special case described in [3].

B.5 Modifications for Single Precision

For single precision usage change all subroutine names beginning with D to begin with S. Change all DOUBLE PRECISION type statements to REAL.

The authors of [1] recommend that the double precision version of this package should be used, except on machines such as the Cray, where single precision arithmetic has precision of about 14.4 decimal places.

C. Examples and Remarks

Example

Define

$$f(t;\mathbf{c}) = c_3 + c_4\cos c_1 t + c_5 \sin c_1 t + c_6 \cos c_2 t + c_7 \sin c_2 t \quad (6)$$

Let NDATA = 30. Generate a data set (t_i, y_i) for $i = 1$, ..., NDATA, by setting $t_i = (i-1)/29$,

$$\hat{\mathbf{c}} = (6, 9, 1, 0.5, 0.4, 0.2, 0.1),$$

and $y_i = f(t_i;\hat{\mathbf{c}}) + \nu_i$, where ν_i is Gaussian noise with mean zero and sample standard deviation 0.001.

The program DRDNLAFU illustrates the use of DNLAFU to find a coefficient vector **c** that best fits this data in the least-squares sense, using the model function of Eq. (6) The computation is started with an initial guess of

$$\mathbf{c}_0 = (5, 10, 0.5, 0.5, 0.5, 0.5, 0.5).$$

The output is shown in ODDNLAFU. The solution vector is

$$\mathbf{c} = (5.99, 9.00, 1.00, 0.502, 0.397, 0.199, 0.100),$$

and the standard deviation of the noise in the data is estimated to be SIGFAC = 0.000986.

Program DRDNLSGU illustrates the solution of this problem treating it as a separable problem and using DNLSGU. To treat this as a separable problem, we relabel the coefficients in Eq. (6) obtaining

$$f(t,\alpha,\beta) = \beta_1 + \beta_2\cos\alpha_1 t + \beta_3 \sin\alpha_1 t + \beta_4 \cos\alpha_2 t + \beta_5 \sin\alpha_2 t \quad (7)$$

This is in the form of Eq. (3) with

$$\begin{array}{lll} \varphi_{i,1} = 1, & \varphi_{i,2} = \cos\alpha_1 t, & \varphi_{i,3} = \sin\alpha_1 t, \\ \varphi_{i,4} = \cos\alpha_2 t, & \varphi_{i,5} = \sin\alpha_2 t & \end{array} \quad (8)$$

Note the setting of the IND(,) array in DRDNLSGU: The 1's in rows 2 and 3 of column 1 indicate that α_1 appears only in $\varphi_{i,2}$ and $\varphi_{i,3}$, while the 1's in rows 4 and 5 of column 2 indicate that α_2 appears only in $\varphi_{i,4}$ and $\varphi_{i,5}$. The zeros in row 6 indicate that there is no $\varphi_{i,6}$ term in the problem, *i.e.*, the problem is of the form of Eq. (3) and not Eq. (4).

Initial comments in subroutine DCALCB explain the storage of partial derivatives in the array DER().

Comparing the results from the DNLAFU and DNLSGU it is seen that the two solutions are of similar accuracy.

These two sample drivers also illustrate the setting of print options by first calling DIVSET to set nominal values into IV() and V(), and then resetting some elements of IV().

As another example of a separable problem, suppose one has data (t_i, y_i), and wishes to fit the **y** data by a rational function of t, say with the residual function defined as

$$r_i(\alpha,\beta) = \frac{\beta_1 + \beta_2 t_i}{1 + \alpha_1 t_i + \alpha_2 t_i^2} - y_i. \quad (9)$$

This can be put into the form of Eq. (3) by defining

$$\varphi_{i,1}(\alpha) = \frac{1}{1 + \alpha_1 t_i + \alpha_2 t_i^2}, \quad (10)$$

and

$$\varphi_{i,2}(\alpha) = \frac{t_i}{1 + \alpha_1 t_i + \alpha_2 t_i^2}. \quad (11)$$

Remarks

1. It is important for success of DNLxxx that the initial guess be as good as possible.
2. DNLxxx only finds a local minimum. Problems may have more than one local minimum, so caution in accepting results is suggested. It may be useful to solve the problem several times using significantly different starting points.
3. Solution of nonlinear least-squares problems is inherently difficult in many cases, and sometimes may require interaction with the user. The internal output available from the subroutine may be useful if one has questions about the performance of the subroutine. It is not uncommon to make mistakes in writing the code for computing partial derivatives. This mistake is likely to cause a return with IV(1) = 8. The subroutine DCKDER of Chapter 8.3 can be used to check the mutual consistency of code for function and Jacobian evaluation.

4. If the different data points have significantly different a priori uncertainties then appropriate row scaling should be used. In the user supplied subroutines (DCALCR, DCALCJ, DCALCA, DCALCB) the i^{th} component of the residual vector and the i^{th} row of the Jacobian matrix should be divided by the a priori standard deviation of the error in y_i.
   To see if the standard deviations introduced in this way are reasonable, one can, after the fit, compute SIGFAC $= \sqrt{2 \times \psi/\mathrm{DOF}}$ where ψ is obtained from V(f), and DOF = NDATA − NC for DNLAxx and NDATA − NA − NB for DNLSxx. SIGFAC is a quantity that could be multiplied times each of the a priori standard deviation values to obtain values that are more consistent with the fit. If SIGFAC is near one it is an indication that the a priori standard deviations were not uniformly unreasonably small or large.

5. If the final residual vector is desired, the user must add code to compute it using the final solution coefficients. If using DNLAxx, this can be done just by calling DCALCR. If using DLNSxx, the user can call DCALCA to evaluate the functions $\varphi_{i,j}(\alpha)$, and then execute code that uses these values, along with the final β's, to compute the residuals using the formulas of Equations 3 or 4 of Section A.

6. With default settings the package generates and updates a scaling vector, **d**, to avoid difficulties that can arise if components of the solution vector are of significantly different magnitudes. In [1] it is reported that in problems where the solution vector components are of about the same order of magnitude the program works better if the vector **d** is held constant. This can be done by setting IV(*dtype*) = 0 and V(*d0init*) = 1.0.

7. To compute the covariance matrix of the solution coefficients see Section B.4.f and IV(*covprt*) of B.4.c. This is only relevant with DNLxxU.

8. We are not describing a reverse communication usage for this package; however, one level below the described user-callable subroutines this package does use reverse communication. If a user needs to embed this package into a larger package and feels reverse communication would be important, he/she could study the way the top level subroutines call the second level ones and mimic this protocol. Specifically DNLAFU and DNLAGU call DRN2G; DNLSGU and DNLSFU call DRNSG; DNLAFB and DNLAGB call DRN2GB; and DNLSFB and DNLSGB call DRNSGB.

D. Functional Description

D.1 Fundamental notation for nonlinear least-squares

The problem is to minimize the function $\psi(\mathbf{c})$ of Eq. (2). Let J denote the NDATA × NC Jacobian matrix of the NDATA-dimensional vector valued function $\mathbf{r}(\mathbf{c})$. Thus the (i,j) element of J is $\partial r_i/\partial c_j$.

The NC-dimensional gradient vector of ψ with respect to the components of **c** is

$$\mathbf{g} = J^t\mathbf{r}$$

Let h_i denote the NC × NC Hessian matrix of second partial derivatives of the i^{th} component of **r** with respect to the components of **c**. Then the NC × NC Hessian matrix of second partial derivatives of ψ with respect to the components of **c** can be written as

$$H = J^tJ + \sum_{i=1}^{NDATA} r_i h_i \tag{12}$$

A necessary condition for ψ to attain a local minimum is that the gradient vector **g** must be zero. Linearizing **g** about a nominal value of **c** gives

$$\mathbf{g}(\mathbf{c}+\delta\mathbf{c}) = \mathbf{g}(\mathbf{c}) + H\delta\mathbf{c} + O(\|\delta\mathbf{c}\|^2). \tag{13}$$

Thus a correction, $\delta\mathbf{c}$, to a current parameter vector, **c**, that tends to move the gradient toward zero is given as the solution of

$$H\delta\mathbf{c} = -\mathbf{g} \tag{14}$$

Eq. (14) is called the full Newton iteration for solving a nonlinear least-squares problem. A different iteration formula, called the Gauss-Newton method can be derived by linearizing $\mathbf{r}(\mathbf{c})$, writing:

$$\mathbf{r}(\mathbf{c}+\delta\mathbf{c}) = \mathbf{r}(\mathbf{c}) + J\delta\mathbf{c} + O(\|\delta\mathbf{c}\|^2). \tag{15}$$

The correction $\delta\mathbf{c}$ that must be added to a current parameter vector, **c**, to make the linearized expression in Eq. (15) close to the zero vector in the least-squares sense is the solution of the linear least-squares problem

$$J\delta\mathbf{c} \simeq -\mathbf{r}. \tag{16}$$

By forming normal equations, we can express this correction vector, $\delta\mathbf{c}$, as the solution of

$$J^tJ\delta\mathbf{c} = -J^t\mathbf{r} \equiv -\mathbf{g}. \tag{17}$$

Note that both of the Eqs. (14) and (17) can be expressed in the general form

$$(J^tJ + S)\delta\mathbf{c} = -\mathbf{g}, \tag{18}$$

where S is $\sum r_i h_i$ for the full Newton method and $S = 0$ for the Gauss-Newton method.

It is known that the correction vector, $\delta\mathbf{c}$, obtained from Eq. (18) with either of the two definitions of S so far mentioned will frequently be too long, especially when the current $\mathbf{c}$ is not close to the solution. Logic can be added to try shorter correction vectors in the same direction; however, a different approach that has been found more effective is to add a term of the form λP to the matrix in Eq. (18), where λ is a nonnegative scalar and P is a positive definite symmetric matrix, sometimes taken as diagonal or the identity matrix. Increasing λ not only decreases the Euclidean norm of $\delta\mathbf{c}$, but also turns $\delta\mathbf{c}$ closer to the direction of local steepest descent. This use of a matrix of the form λP is associated with the names of Levenberg and Marquardt.

In the original Marquardt approach λ was the primary control parameter. Subsequent "trust-region" algorithms have set target values for $\|\delta\mathbf{c}\|$, or more precisely for $\|D\delta\mathbf{c}\|$ using a scaling matrix D, and determined λ to achieve that target value.

D.2 The NL2SOL algorithm

The present package has evolved from the NL2SOL package of [1]. Additional details of the underlying techniques are given in [4]. A trust-region method is used with $P = D^2$, where D is a diagonal matrix with positive diagonal elements. Thus Eq. (18) is replaced by

$$(J^tJ + S + \lambda D^2)\delta\mathbf{c} = -\mathbf{g}. \tag{19}$$

The full Newton method has the advantage over the Gauss-Newton method of providing a more complete model of the nonlinear problem and ultimately having a quadratic rate of convergence when close enough to the solution. It has the disadvantage of requiring computation of a large number of second derivative terms. The authors of [1] developed a reasonably inexpensive method of sequentially constructing a matrix, S, that has properties in common with the true S matrix of the full Newton method in some directions of NC-dimensional parameter space. They found that using this approximate S in Eq. (19) was sometimes better and sometimes worse than using $S = 0$, the latter being a Gauss-Newton-Marquardt method.

The complete algorithm developed in [1] updates the approximate S matrix at each iteration and then does tests to decide whether to use this S or $S = 0$ in Eq. (19) to determine the next $\delta\mathbf{c}$. The authors found that their strategy of choosing between the approximate S and $S = 0$ at each iteration led to better performance over many test cases than the exclusive use of either the approximate S or $S = 0$. In the detailed output selected by IV(*outlev*) the method used is indicated by "S" when the approximate S matrix is used and "G" (for Gauss-Newton-Marquardt) when $S = 0$ is used.

D.3 Specializing for the separable problem

Refer to Eqs. (2), (3), and (4). Let Φ denote the NDATA $\times$ NB matrix with components $\varphi_{i,j}$. Let $\mathbf{z}$ denote an NDATA dimensional vector equal to $\mathbf{y}$ in Eq. (3), and with components $z_i = y_i - \varphi_{i,\mathrm{NB}+1}$ in Eq. (4). Note that Φ is a function of α, and so also is $\mathbf{z}$ when using Eq. (4).

The objective function of Eq. (2) can be expressed as

$$\begin{aligned}\psi(\alpha,\beta) &= \frac{1}{2}\min_{\alpha,\beta}\|\mathbf{r}(\alpha,\beta)\|^2 \\ &= \frac{1}{2}\min_{\alpha}\left\{\min_{\beta}\|\Phi\beta - \mathbf{z}\|^2\right\}.\end{aligned} \tag{20}$$

For fixed α the minimizing vector β can be expressed as $\beta = \Phi^+\mathbf{z}$, where Φ^+ denotes the pseudoinverse of Φ. Thus, the optimal residual vector associated with a fixed α can be expressed as

$$\mathbf{r}(\alpha) = \Phi\Phi^+\mathbf{z} - \mathbf{z} = (\Phi\Phi^+ - I)\mathbf{z} \tag{21}$$

Then Eq. (20) can be rewritten as

$$\psi(\alpha) = \frac{1}{2}\min_{\alpha}\|\mathbf{r}(\alpha)\|^2, \tag{22}$$

where $\mathbf{r}(\alpha)$ is defined by Eq. (21)

Thus a problem that appeared to depend on an NA-vector α and an NB-vector β has been reformulated to depend only on α. The vector β can be computed from α as needed.

Reference [5] may have been the first paper to explicitly present this idea for reducing the effective dimensionality of a nonlinear least-squares problem. More algorithmic details, particularly the transformations needed to compute partial derivatives of ψ with respect to α from the given partial derivatives of the $\varphi_{i,j}$'s with respect to α, are given in [6] and [7]. The approach of this package is based particularly on [7], with the transformed NA-dimensional problem being solved using the NL2SOL algorithm of [1].

D.4 D, d, and parameter scaling

The algorithm uses a vector, $\mathbf{d}$, of positive scaling values to compensate for possible disparate scaling of the solution parameters at various points in the computation, particularly in convergence tests and in constructing the D matrix of Eq. (19). Thus, D is defined to be the diagonal matrix with the components of the vector $\mathbf{d}$ on its diagonal.

For DNLAxx the dimension of $\mathbf{d}$ and the order of D is NC, and the scaled coefficient vector is $D\mathbf{c}$. For DNLSxx the dimension of $\mathbf{d}$ and the order of D is NA, and the scaled (nonlinear) coefficient vector is $D\alpha$. The

scaled relative distance between two vectors is defined by Eq. (5), page 9.

If the magnitudes of the individual solution components, *i.e.*, the c_j's or α_j's, are fairly well known a priori, it is suggested that the components of **d** be set to the reciprocal of these magnitudes and held constant throughout the solution process. Otherwise it is suggested that the algorithm be permitted to control **d**. See the specification of IV(*dtype*), page 10 for specifications on how to exercise this option.

D.5 Covariance Matrix

Reference [1] refers to [2] for discussion of three different matrices that might be regarded as the covariance matrix for the solution parameters of a nonlinear least-squares problem. See the specification of IV(*covreq*) in Section B.4.f for further information. The default choice, |IV(*covreq*)| = 1, is recommended in [1], whereas |IV(*covreq*)| = 3 is recommended in [8]. When the Hessian matrix, H, is needed for computing the covariance matrix, H is approximated by finite differences.

D.6 Influence Coefficients (Regression Diagnostics)

A method treated in the statistical literature for judging the relative influence of different data points on a fit is the *leave one out* analysis. The idea is to repeat the fit NDATA times, with the i^{th} fit done using all except the i^{th} data point. Approaches to producing this type of information without repeating the fit NDATA times are presented in pp. 996–997 of [9]. This feature was not in NL2SOL ([1]) but one of the methods of [9] is implemented in the present package.

Let **x** denote the solution vector (**c** or α), and let ψ_0 be the value of the objective function at **x**. For i in [1, NDATA] let $\mathbf{g}^{(i)}$ and $H^{(i)}$ denote the gradient vector and Hessian matrix, respectively, computed using **x** and omitting the contribution due to the i^{th} data point. Define

$$\gamma_i = \left(\mathbf{g}^{(i)t} H^{(i)-1} \mathbf{g}^{(i)}\right)^{1/2}. \tag{23}$$

Note that γ_i^2 is a first order estimate of the amount by which the objective function ψ would be reduced from ψ_0 if the problem were solved omitting the i^{th} data point. In tests of this feature the values of γ_i have been found to be from 0.6 to 1.2 times the true value. For a more reliable *leave one out* analysis we suggest actually computing the NDATA different solutions, starting each solution from the solution of the full problem.

If requested by the setting of IV(*rdreq*) = 2 or 3, the package computes γ_i, for i = 1, ..., NDATA, and stores these values in V() starting at V(IV(*regd*)). Printing of the γ's is controlled by IV(*covprt*).

If H is indefinite, the γ's will not be computed and IV(*regd*) will be set to −1. If the γ's are not computed for any other reason, IV(*regd*) will be set to zero. If an individual $H^{(i)}$ is indefinite the corresponding γ_i will be set to −1.0.

References

1. John E. Dennis, Jr., David M. Gay, and Roy E. Welsch, *An adaptive nonlinear least-squares algorithm*, **ACM Trans. on Math. Software 7**, 3 (Sept. 1981) 348–368. Also, Algorithm 573, NL2SOL, pp. 369–383.

2. Y. Bard, **Nonlinear Parameter Estimation**, Academic Press, New York (1974) 341 pages.

3. David M. Gay, *Computing optimal locally constrained steps*, **SIAM J. on Scientific Computing 2** (1981) 186–197.

4. 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.

5. William H. Lawton and Edward A. Sylvestre, *Elimination of linear parameters in nonlinear regression*, **Technometrics 13**, 3 (Aug. 1971) 461–467.

6. Gene H. Golub and Victor Pereyra, *The differentiation of pseudo-inverse and nonlinear least-squares problems whose variables separate*, **SIAM J. on Numer. Anal. 10** (1973) 413–432.

7. Linda C. Kaufman, *A variable projection method for solving separable nonlinear least squares problems*, **BIT 15** (1975) 49–57.

8. J. R. Donaldson and R. B. Schnabel, *Computational experience with confidence regions and confidence intervals for nonlinear least squares*, **Technometrics 29** (1987) 67–82.

9. David M. Gay and Roy E. Welsh, *Maximum likelihood and quasi-likelihood for nonlinear exponential family regression models*, **J. Amer. Stat. Assn. 83**, 404 (Dec. 1988) 990–998.

10. Thomas F. Coleman, Burton S. Garbow, and Jorge J. Moré, *Software for estimating sparse Jacobian matrices*, **ACM Trans. on Math. Software 10**, 3 (Sept. 1984) 329–345. Also, Algorithm 618, pp. 346–347.

11. David M. Gay, **Usage Summary for Selected Optimization Routines**. Internal Bell Laboratory Document, AT&T (May 1984).

E. Error Procedures and Restrictions

On all returns, successful or not, the reason for the return is indicated by IV(1). See Section B.4.a for the interpretation of these values.

After a return with IV(1) $\leq$ 11, it is possible to restart, *i.e.*, to change some of the IV() and V() input values described in Sections B.4.c through B.4.i, and continue the algorithm from the point where it was interrupted. IV(1) should not be changed.

This package does not use the MATH77 error printing subroutines. Error conditions are reported by direct printing to the I/O unit selected by IV(*prunit*) if IV(*prunit*) $>$ 0. A message will be printed for successful returns if IV(*prunit*) $>$ 0 and at least one option to print results has been selected. If IV(*prunit*) = 0, all printing of both results and error messages is suppressed.

Subroutines containing WRITE statements, and the number of WRITE statements in each, are DITSUM,32; DN2CVP,11; DN2RDP,1; DPARCK,16; and DS7CPR,1.

F. Supporting Information

The source language is ANSI Fortran 77.

The original code corresponding to DNLAGU/DNLAFU was NL2SOL/NL2SNO, developed in 1976–1980, partially supported by NSF grants MCS–7600324, DCR75–10143, 76–14311DSS, MCS76–11989, and MCS–7906671, and published in [1]. A precursor of the separable code, DNLSGU, was written by Linda Kaufman in 1977; see [7]. The code in the file IDSM is from [10].

Additional development, resulting in the eight main user-callable subroutines described here was done by David Gay and Linda Kaufman at AT&T Bell Laboratories, Murray Hill N.J., from 1980 through 1990. This code was developed for use in the PORT library that is used internally at AT&T and is also leased to other organizations. In addition the code was placed in the `/netlib/port` directory of `netlib.att.com` on the Internet to make it publicly available. The code was downloaded from there to JPL by C. L. Lawson in February 1990. Changes were made to this code to make it more consistent with the style of codes in the JPL MATH77 library.

This writeup by C. L. Lawson, JPL, is based on [1], [11], comment lines in the codes, and very helpful personal communication with David Gay.

Entry	Required Files
DIVSET	AMACH, DIVSET
DNLAFB	AMACH, DIVSET, DNLAFB, DRN2GB, I7COPY
DNLAFU	AMACH, DIVSET, DNLAFU, DRN2G
DNLAGB	AMACH, DIVSET, DNLAGB, DRN2GB, I7COPY
DNLAGU	AMACH, DIVSET, DNLAGU, DRN2G
DNLSFB	AMACH, DIVSET, DNLSFB, DQ7RFH, DRN2GB, DRNSGB, I7COPY, IDSM
DNLSFU	AMACH, DIVSET, DNLSFU, DQ7RFH, DRN2G, DRNSG, IDSM
DNLSGB	AMACH, DIVSET, DNLSGB, DQ7RFH, DRN2GB, DRNSGB, I7COPY
DNLSGU	AMACH, DIVSET, DNLSGU, DQ7RFH, DRN2G, DRNSG
SIVSET	AMACH, SIVSET
SNLAFB	AMACH, I7COPY, SIVSET, SNLAFB, SRN2GB
SNLAFU	AMACH, SIVSET, SNLAFU, SRN2G
SNLAGB	AMACH, I7COPY, SIVSET, SNLAGB, SRN2GB
SNLAGU	AMACH, SIVSET, SNLAGU, SRN2G
SNLSFB	AMACH, I7COPY, IDSM, SIVSET, SNLSFB, SQ7RFH, SRN2GB, SRNSGB
SNLSFU	AMACH, IDSM, SIVSET, SNLSFU, SQ7RFH, SRN2G, SRNSG
SNLSGB	AMACH, I7COPY, SIVSET, SNLSGB, SQ7RFH, SRN2GB, SRNSGB
SNLSGU	AMACH, SIVSET, SNLSGU, SQ7RFH, SRN2G, SRNSG

DRDNLAFU

```
c      program DRDNLAFU
c>> 1994-11-02 DRDNLAFU Krogh  Changes to use M77CON
c>> 1994-09-14 DRDNLAFU CLL Set IV(OUTLEV) = 0 for comparing output.
c>> 1992-02-03 CLL @ JPL
c>> 1990-07-02 CLL @ JPL
c>> 1990-06-27 CLL @ JPL
c>> 1990-06-14 CLL @ JPL
c>> 1990-03-28 CLL @ JPL
c      Demo driver for DNLAFU. A variant of the nonlinear LS code NL2SOL.
c      DNLAFU requires function values only.
c     ------------------------------------------------------------------
c--D replaces "?": DR?NLAFU, ?NLAFU, ?CALCR, ?IVSET
c     ------------------------------------------------------------------
      external DCALCR
      integer  LIV, LV, MC, MDATA, NC, NDATA
      parameter(MDATA = 30, MC = 7)
      parameter(LIV = 82 + MC)
      parameter(LV = 105 + MC*(MDATA + 2*MC + 17) + 2*MDATA)
      integer IV(LIV)
      integer F, COVPRT, OUTLEV, SOLPRT, STATPR, X0PRT
      parameter(F=10)
      parameter(COVPRT=14, OUTLEV=19, SOLPRT=22, STATPR=23, X0PRT=24)
      double precision COEF(MC), DOF, V(LV)
c     ------------------------------------------------------------------
      NDATA = MDATA
      NC = MC
      COEF(1) =     5.0d0
      COEF(2) =    10.0d0
      COEF(3) =     0.5d0
      COEF(4) =     0.5d0
      COEF(5) =     0.5d0
      COEF(6) =     0.5d0
      COEF(7) =     0.5d0
      IV(1) = 0

      print'(1x,a)',
     * 'Program DRDNLAFU.. Demo driver for DNLAFU.',
     * '   A variant of NL2SOL.',
     * '   DNLAFU requires function values but not the Jacobian.',
     * ' ',
     * 'Sample problem is a nonlinear curve fit to data.',
     * 'Model function is C3 + C4 * cos(C1*t) + C5 * sin(C1*t) +',
     * '                   C6 * cos(C2*t) + C7 * sin(C2*t) + Noise',
     * 'Data generated using',
     * '(C1, ..., C7) = (6, 9, 1, 0.5, 0.4, 0.2, 0.1)',
     * 'and Gaussian noise with mean 0 and',
     * 'sample standard deviation 0.001',
     * ' '

      call DIVSET(1, IV, LIV, LV, V)
      IV( X0PRT) = 1
      IV(OUTLEV) = 0
      IV(STATPR) = 1
      IV(SOLPRT) = 1
      IV(COVPRT) = 1
```

```
      call DNLAFU(NDATA, NC, COEF, DCALCR, IV, LIV, LV, V)

      DOF = max(NDATA - NC, 1)
      print '(1x/1x,a,g12.4)',
     *     'SIGFAC: sqrt((2 * V(F))/DOF) =',
     *              sqrt(2.0d0 * V(F)/DOF)
      stop
      end
c     ==================================================================
      subroutine DCALCR(NDATA, NC, C, NCOUNT, RVEC)
c     Function evaluation to test nonlinear least squares computation.
c     ------------------------------------------------------------------
      integer I, MDATA, NCOUNT, NDATA, NC
      parameter(MDATA = 30)
      double precision C(NC), DEL, RVEC(NDATA), T, YDATA(MDATA)
      data YDATA /
     *     1.700641d0, 1.793512d0, 1.838309d0, 1.838416d0, 1.792204d0,
     *     1.700501d0, 1.579804d0, 1.426268d0, 1.260724d0, 1.084901d0,
     *     0.917094d0, 0.761920d0, 0.627304d0, 0.522146d0, 0.446645d0,
     *     0.404920d0, 0.392033d0, 0.409622d0, 0.453045d0, 0.510765d0,
     *     0.584554d0, 0.663109d0, 0.747613d0, 0.829439d0, 0.908496d0,
     *     0.983178d0, 1.051046d0, 1.114072d0, 1.171746d0, 1.227823d0/
c     ------------------------------------------------------------------
      T = 0.0D0
      DEL = 1.0D0 / 29.0D0
      do 10 I = 1,NDATA
         RVEC(I) = C(3) + C(4)*cos(C(1)*T) + C(5)*sin(C(1)*T) +
     *                    C(6)*cos(C(2)*T) + C(7)*sin(C(2)*T) -
     *                    YDATA(I)
         T = T + DEL
   10 continue
      return
      end
```

ODDNLAFU

```
Program DRDNLAFU.. Demo driver for DNLAFU.
   A variant of NL2SOL.
   DNLAFU requires function values but not the Jacobian.

Sample problem is a nonlinear curve fit to data.
Model function is C3 + C4 * cos(C1*t) + C5 * sin(C1*t) +
                    C6 * cos(C2*t) + C7 * sin(C2*t) + Noise
Data generated using
(C1, ..., C7) = (6, 9, 1, 0.5, 0.4, 0.2, 0.1)
and Gaussian noise with mean 0 and
sample standard deviation 0.001


     I      INITIAL X(I)        D(I)

     1       5.00000          0.621
     2       10.0000          0.609
     3      0.500000           1.00
     4      0.500000           1.00
     5      0.500000           1.00
     6      0.500000           1.00
     7      0.500000          0.997

***** RELATIVE FUNCTION CONVERGENCE *****

FUNCTION      0.111899E-04   RELDX          0.166E-07
FUNC. EVALS       11         GRAD. EVALS       70
PRELDF        0.377E-11      NPRELDF        0.377E-11

     I       FINAL X(I)         D(I)           G(I)

     1       5.99129          0.621          0.540E-11
     2       8.99554          0.609         -0.546E-11
     3       1.00057           1.00          0.631E-11
     4      0.501649           1.00          0.508E-10
     5      0.396734           1.00          0.190E-10
     6      0.198612           1.00         -0.326E-10
     7      0.100243           1.00         -0.258E-10

  36 EXTRA FUNC. EVALS FOR COVARIANCE AND DIAGNOSTICS.

RECIPROCAL CONDITION OF F.D. HESSIAN = AT MOST  0.22E-04

 COVARIANCE = VARFAC * H**-1 * (J**T * J) * H**-1
WHERE H = F.D. HESSIAN

ROW  1     0.304E-03
ROW  2    -0.498E-03   0.106E-02
ROW  3    -0.240E-04   0.429E-04   0.199E-05
ROW  4    -0.771E-04   0.142E-03   0.635E-05   0.208E-04
ROW  5     0.143E-03  -0.251E-03  -0.116E-04  -0.374E-04   0.688E-04

ROW  6     0.916E-04  -0.174E-03  -0.760E-05  -0.248E-04   0.449E-04
           0.301E-04
ROW  7    -0.648E-04   0.119E-03   0.526E-05   0.171E-04  -0.314E-04
          -0.208E-04   0.147E-04

SIGFAC: sqrt((2 * V(F))/DOF) =  0.9864E-03
```

DRDNLSGU

```
c      program DRDNLSGU
c>> 2001-05-24 DRDNLSGU Krogh Minor change for making .f90 version.
c>> 1997-06-18 DRDNLSGU Krogh  Changes to improve C portability.
c>> 1994-11-02 DRDNLSGU Krogh  Changes to use M77CON
c>> 1994-09-14 DRDNLSGU CLL Set IV(OUTLEV) = 0 for comparing output.
c>> 1992-04-13 CLL Rename and reorder common block [D/S]KEY.
c>> 1992-02-03 CLL @ JPL
c>> 1990-07-02 CLL @ JPL
c>> 1990-06-27 CLL @ JPL
c>> 1990-04-05 CLL @ JPL
c>> 1990-03-29 CLL @ JPL
c      Demo driver for DNLSGU. A variant of the nonlinear LS code NL2SOL.
c      DNLSGU solves the "separable" problem.
c      DNLSGU requires values of the function and the Jacobian matrix.
c      Note:  MDER is the number of ones in the array IND().
c      ------------------------------------------------------------------
c--D replaces "?": DR?NLSGU, ?NLSGU, ?CALCA, ?CALCB, ?IVSET, ?KEY
c      ------------------------------------------------------------------
       external DCALCA, DCALCB
       integer  ITERM,IVAR,LIV,LV,MA,MB,MDATA,MDER,MLEN,NA,NB,NDATA
       parameter(MDATA = 30, MA = 2, MB = 5)
       parameter(MDER = 4)
       parameter(LIV = 115 + MA + MB + 2*MDER)
       parameter(MLEN = (MB+MA)*(MDATA+MB+MA+1))
       parameter(LV = 105 + MDATA*(MB+MDER+3) +
     *                MLEN + (MB*(MB+3))/2 + MA*(2*MA+17))
       integer IND(MB+1,MA), IV(LIV)
       integer F, COVPRT, OUTLEV, SOLPRT, STATPR, X0PRT
       parameter(F=10)
       parameter(COVPRT=14, OUTLEV=19, SOLPRT=22, STATPR=23, X0PRT=24)
       double precision ALF(MA), BET(MB), DOF, V(LV), YDATA(MDATA)
       data ((IND(ITERM,IVAR),IVAR = 1,2),ITERM = 1,6)/
     *      0, 0,
     *      1, 0,
     *      1, 0,
     *      0, 1,
     *      0, 1,
     *      0, 0/
       data YDATA /
     *      1.700641d0, 1.793512d0, 1.838309d0, 1.838416d0, 1.792204d0,
     *      1.700501d0, 1.579804d0, 1.426268d0, 1.260724d0, 1.084901d0,
     *      0.917094d0, 0.761920d0, 0.627304d0, 0.522146d0, 0.446645d0,
     *      0.404920d0, 0.392033d0, 0.409622d0, 0.453045d0, 0.510765d0,
     *      0.584554d0, 0.663109d0, 0.747613d0, 0.829439d0, 0.908496d0,
     *      0.983178d0, 1.051046d0, 1.114072d0, 1.171746d0, 1.227823d0/
c      ------------------------------------------------------------------
       NDATA = MDATA
       NA = MA
       NB = MB
       ALF(1) =    5.0d0
       ALF(2) =   10.0d0
       IV(1) = 0

       print'(1x,a)',
     * 'Program DRDNLSGU.. Demo driver for DNLSGU.',
     * '   A variant of NL2SOL.',
```

```
     * '   DNLSGU handles the Separable problem.',
     * '   DNLSGU requires values of the function and the Jacobian.',
     * ' ',
     * 'Sample problem is a nonlinear curve fit to data.',
     * 'Model function is B1 + B2 * cos(A1*t) + B3 * sin(A1*t) +',
     * '                    B4 * cos(A2*t) + B5 * sin(A2*t) + Noise',
     * 'Data generated using',
     * '(A1, A2, B1, ..., B5) = (6, 9, 1, 0.5, 0.4, 0.2, 0.1)',
     * 'and Gaussian noise with mean 0 and',
     * 'sample standard deviation 0.001',
     * ' '

      call DIVSET(1, IV, LIV, LV, V)
      IV( X0PRT) = 1
      IV(OUTLEV) = 0
      IV(STATPR) = 1
      IV(SOLPRT) = 1
      IV(COVPRT) = 1

      call DNLSGU(NDATA, NA, NB, ALF, BET, YDATA, DCALCA, DCALCB,
     *          IND, NB+1, IV, LIV, LV, V)

      DOF = max(NDATA - NA - NB, 1)
      print '(1x/1x,a,g12.4)',
     *     'SIGFAC: sqrt((2 * V(F))/DOF) =',
     *              sqrt(2.0d0 * V(F)/DOF)
      stop
      end
c     ================================================================
      subroutine DCALCA(NDATA, NA, NB, ALF, NCOUNT, PHI)
c     Test case for separable nonlinear least squares computation.
c     Computes MDATA x NB matrix PHI as a function of the
c     nonlinear parameters ALF().
c     For J .le. NB the (I,J) term of PHI is the coefficient of the
c     linear coefficient B(J) in row I of the model.
c     In this example the model does not have a term that is not
c     multiplied by a linear coefficient.  If such a term is present
c     then PHI must have an (NB+1)st column to hold this term.
c     This code Illustrates saving results in common between DCALCA and
c     DCALCB to avoid recalculation of common subexpressions.
c     ----------------------------------------------------------------
      common/DKEY/C1, C2, S1, S2, KEY
      save  /DKEY/
      integer I, KEY, MDATA, NA, NB, NCOUNT, NDATA
      parameter(MDATA = 30)
      double precision ALF(NA), C1(MDATA), C2(MDATA)
      double precision DEL, PHI(NDATA,NB)
      double precision S1(MDATA), S2(MDATA), T
c     ----------------------------------------------------------------
      T = 0.0D0
      DEL = 1.0D0 / 29.0D0
      KEY = NCOUNT
      do 10 I = 1,NDATA
         C1(I) = cos(ALF(1)*T)
         S1(I) = sin(ALF(1)*T)
         C2(I) = cos(ALF(2)*T)
         S2(I) = sin(ALF(2)*T)
         PHI(I,1) = 1.0D0
         PHI(I,2) = C1(I)
```

```
      PHI(I,3) = S1(I)
      PHI(I,4) = C2(I)
      PHI(I,5) = S2(I)
      T = T + DEL
   10 continue
      return
      end
c     ==================================================================
      subroutine DCALCB(NDATA, NA, NB, ALF, NCOUNT, DER)
c     Test case for separable nonlinear least squares computation.
c     Computes the NDATA x NDER matrix DER.  Here NDER is the number of
c     ones in the indicator array IND().  The columns of DER correspond
c     to nonzero entries of IND() traversed columnwise.
c     In this example NDER is 4 and the correspondence is as follows:
c        Col of DER:          1       2       3       4
c        Element of IND(): (2,1)   (3,1)   (4,2)   (5,2)
c     In this example Row I of DER will be set to contain the values
c     of the four partial derivatives:
c        Partial of PHI(I,2) with respect to ALP(1)
c        Partial of PHI(I,3) with respect to ALP(1)
c        Partial of PHI(I,4) with respect to ALP(2)
c        Partial of PHI(I,5) with respect to ALP(2)
c     ------------------------------------------------------------------
      common/DKEY/C1, C2, S1, S2, KEY
      save /DKEY/
      integer I, KEY, MDATA, NCOUNT, NDATA, NA, NB, NDER
      parameter(MDATA = 30, NDER = 4)
      double precision ALF(NA), C1(MDATA), C2(MDATA)
      double precision DEL, DER(NDATA,NDER), S1(MDATA), S2(MDATA), T
c     ------------------------------------------------------------------
      T = 0.0D0
      DEL = 1.0D0 / 29.0D0
      if(NCOUNT .eq. KEY) then
         do 10 I = 1,NDATA
            DER(I,1) = -S1(I)*T
            DER(I,2) =  C1(I)*T
            DER(I,3) = -S2(I)*T
            DER(I,4) =  C2(I)*T
            T = T + DEL
   10    continue
      else
         do 20 I = 1,NDATA
            DER(I,1) = -sin(ALF(1)*T)*T
            DER(I,2) =  cos(ALF(1)*T)*T
            DER(I,3) = -sin(ALF(2)*T)*T
            DER(I,4) =  cos(ALF(2)*T)*T
            T = T + DEL
   20    continue
      endif
      return
      end
```

ODDNLSGU

```
Program DRDNLSGU.. Demo driver for DNLSGU.
   A variant of NL2SOL.
   DNLSGU handles the Separable problem.
   DNLSGU requires values of the function and the Jacobian.

Sample problem is a nonlinear curve fit to data.
Model function is B1 + B2 * cos(A1*t) + B3 * sin(A1*t) +
                    B4 * cos(A2*t) + B5 * sin(A2*t) + Noise
Data generated using
(A1, A2, B1, ..., B5) = (6, 9, 1, 0.5, 0.4, 0.2, 0.1)
and Gaussian noise with mean 0 and
sample standard deviation 0.001


    I      INITIAL X(I)         D(I)

    1       5.00000          0.685E-01
    2       10.0000          0.278E-01

***** COEFFICIENT AND RELATIVE FUNCTION CONVERGENCE *****

FUNCTION      0.111899E-04   RELDX         0.235E-08
FUNC. EVALS       6          GRAD. EVALS       5
PRELDF        0.102E-12      NPRELDF       0.102E-12

    I       FINAL X(I)         D(I)           G(I)

    1      5.99129          0.703E-01      0.531E-10
    2      8.99554          0.419E-01     -0.194E-10

LINEAR PARAMETERS...

    1      1.00057
    2     0.501649
    3     0.396734
    4     0.198612
    5     0.100243

   3 EXTRA FUNC. EVALS FOR COVARIANCE AND DIAGNOSTICS.
   3 EXTRA GRAD. EVALS FOR COVARIANCE AND DIAGNOSTICS.

RECIPROCAL CONDITION OF F.D. HESSIAN = AT MOST  0.22E-04

 COVARIANCE = VARFAC * H**-1 * (J**T * J) * H**-1
WHERE H = F.D. HESSIAN

ROW  1      0.299E-03
ROW  2     -0.487E-03   0.104E-02
ROW  3     -0.235E-04   0.420E-04   0.195E-05
ROW  4     -0.757E-04   0.139E-03   0.623E-05   0.203E-04
ROW  5      0.141E-03  -0.245E-03  -0.113E-04  -0.367E-04   0.676E-04

ROW  6      0.899E-04  -0.170E-03  -0.745E-05  -0.243E-04   0.440E-04
            0.295E-04
ROW  7     -0.636E-04   0.117E-03   0.516E-05   0.168E-04  -0.308E-04
           -0.204E-04   0.145E-04

SIGFAC: sqrt((2 * V(F))/DOF) =  0.9864E-03
```