AT&T Bell Laboratories
Murray Hill, NJ 07974

Computing Science Technical Report No. 153

Usage Summary for Selected Optimization Routines

David M. Gay

October 1990

USAGE SUMMARY FOR SELECTED OPTIMIZATION ROUTINES†

David M. Gay

CONTENTS

† This is a reprint, with minor corrections noted in footnotes, of the *USAGE SUMMARY FOR SELECTED OPTIMIZATION ROUTINES* that appears in the Optimization chapter of the *PORT Mathematical Subroutine Library* manual of 1984.

1. Introduction

Several PORT optimization routines have a common structure and user interface, all similar to that described in [2]. All controls and tolerances — and all scratch storage — used by these routines are contained in two arrays: IV for integer values, V for floating-point values (REAL in the single-precision versions, DOUBLE PRECISION in the double-precision versions); this usage summary does not address the simplified versions of these routines (e.g. SN2F and SMNF) that allocate these arrays for you from the PORT stack. The discussions below use symbolic subscripts, such as IV(MXITER), in describing various components of IV and V. Numerical values for these symbolic subscripts appear in §18 and are also given when a component is discussed, as in ‘‘IV(MXITER) = IV(18) is the maximum number of iterations allowed.’’ (One exception is the first component of IV, IV(1), which on input says what kind of call this is and on output contains a return code. It is always called IV(1).)

This usage summary is written to encompass some routines that will not be included in the initial release of PORT 3, such as routines for nonlinear Poisson, logistic, and robust regression and versions of the various optimization routines that handle general linear constraints. This summary is designed to remain valid when such routines are added to the library.

1a. Notation

Given a function f of p variables, the optimization routines attempt to find a p-vector x^* that minimizes $f(x)$. Various constraints may be imposed on x: none, simple bounds of the form

$$\underline{b}^x \le x \le \overline{b}^x \tag{1.1}$$

(where $\underline{b}^x$ and $\overline{b}^x$ are vectors and the inequalities are understood componentwise), or general linear constraints of the form

$$\underline{b}^c \le Cx \le \overline{b}^c \tag{1.2}$$

(where $\underline{b}^c$ and $\overline{b}^c$ are vectors and C is a matrix). The gradient of f at x (vector of first partial derivatives of f) will be denoted by $\nabla f(x)$, and the Hessian of f at x (matrix of second partial derivatives of f) will be denoted by $\nabla^2 f(x)$.

If z is a vector of m components, $z = (z_1, z_2, \cdots, z_m)^T$, then $\|z\|$ denotes its Euclidean norm (2–norm),

$$\|z\| = \left[\sum_{i=1}^{m} z_i^2 \right]^{1/2} . \tag{1.3}$$

MACHEP, which appears in some expressions for default V values, denotes the unit roundoff on the current machine (the value returned by the PORT function `R1MACH(4)` or `D1MACH(4)`).

1b. Caveat

Unless f is convex, the PORT optimization routines may only find a local minimum, even when ‘‘better’’ minima exist. When you think this is a danger, you may wish to try several starting points.

1c. Forward and reverse communication

The optimization routines addressed by this usage summary have at least two levels:

- forward-communication routine
- reverse-communication iteration driver

Forward-communication routines learn about $f(x)$ in the conventional way: you give them a subroutine they can call to compute $f(x)$. Reverse-communication drivers, on the other hand, return to their caller (e.g. your main program) whenever they need to know $f(x)$ at a new x. The calling routine must then compute the necessary information (e.g. $f(x)$ itself or, for some regression routines, a residual vector) and call the reverse-communication driver again, passing it the information it wants. Usually it is easier to use a forward-communication optimization routine, but sometimes it is simpler to call a reverse-communication driver, e.g. when writing a subroutine that computes $f(x)$ is inconvenient.

Both the forward- and the reverse-communication versions of the optimization routines receive IV and V as parameters. Parameters to the reverse-communication drivers also include x, as well as either the values of $f(x)$ and perhaps $\nabla f(x)$ or information sufficient to compute these values (for regression routines). The driver returns with IV(1) = 1 when it wants to have f evaluated at the current x and with IV(1) = 2 when it wants ∇f evaluated. Some drivers have other possible returns, such as IV(1) = –1 or –2; see the appropriate PORT reference sheet for details.

2. Overriding defaults

As explained above, input controls and tolerances are passed in two arrays, IV and V. You may run with all IV and V input components at their default values by setting IV(1) to 0 before calling the optimization routine. (Default values are described in various sections below.) Sometimes you may need to relax the default stopping tolerances, turn off some of the default printing, or otherwise turn the input knobs. To do so, you first call subroutine IVSET (for single precision, DIVSET for double) to supply IV and V with default values. You then assign nondefault values to appropriate components of IV and V. Finally, you call the relevant optimization routine, passing IV and V to it. The calling sequence for IVSET is

```
CALL IVSET(KIND, IV, LIV, LV, V)
```

where `KIND` is an integer, chosen as in Table 1 below — and specified on the relevant PORT reference sheet as well.

`KIND`	Kind of optimization
1	unconstrained or simply bounded regression
2	unconstrained or simply bounded general optimization
3	regression with general linear constraints
4	general optimization with general linear constraints

Table 1

The integer parameters LIV and LV give the lengths of the IV and V arrays you are providing. The PORT reference sheet for the relevant optimization routine gives minimum acceptable values for LIV and LV, values that are functions of the problem dimensions. It is usually simplest to be liberal in choosing LIV and LV — to guess (or compute) overestimates of their minimum values; so long as you do not run out of storage, making LIV and LV larger than necessary should cause no harm. If you make LIV or LV too small (but make LIV at least 21, so that the printing unit number, IV(PRUNIT) = IV(21) can be stored), then a message giving the minimum acceptable values of LIV and LV will be printed. Also, (if LIV is at least 45) the optimization routine will store these minimum LIV and LV values in IV(LASTIV) = IV(44) and IV(LASTV) = IV(45) respectively.

Example: To turn off all printing when calling an unconstrained general optimization routine (e.g., MNF or MNG), execute

```
CALL IVSET(2, IV, LIV, LV, V)
IV(19) = 0
```

See §6a for more information on print controls.

3. Return codes

When the optimization routines return, IV(1) contains a return code (a number that indicates how the routine fared). The desirable return codes are 3, 4, 5, and sometimes 6. The meanings of these return codes are sketched in the list of return codes below and described in more detail in §5. Return codes include:

Favorable returns:

1–2 — impossible: these are input IV(1) values only (see §1c).

3 — X-convergence: the current iterate appears to be a scaled distance (see §5) of at most V(XCTOL) = V(33) from a locally optimal point.

4 — relative function convergence: the current objective function value $f(x)$ appears to differ from a locally optimal value by at most $|f(x)| \cdot \mathrm{V(RFCTOL)} = |f(x)| \cdot \mathrm{V}(32)$.

5 — both X- and relative function convergence (3 and 4 combined).

6 — absolute function convergence: $|f(x)| < \mathrm{V(AFCTOL)}$ = V(31). This test is only of interest in problems where $f(x) = 0$ means a ''perfect fit'', such as nonlinear least-squares problems.

Error returns from which restarts (§13) are possible:

7 — singular convergence: *x* may have too many free components. See §5.

8 — false convergence: the gradient $\nabla f(x)$ may be computed incorrectly, the other stopping tolerances may be too tight, or either *f* or ∇f may be discontinuous near the current iterate *x*.

9 — function evaluation limit: no convergence after IV(MXFCAL) = IV(17) evaluations of $f(x)$.

10 — iteration limit: no convergence after IV(MXITER) = IV(18) iterations.

11 — STOPX returned .TRUE.: you supplied a system-dependent STOPX (see §12) routine and hit the BREAK key.

12–13 — impossible: these are input IV(1) values only. (12 means allocate storage within IV and V and start the algorithm; this is the default IV(1) value supplied by [D]IVSET. 13 means just allocate storage and return. See §4a for an example.)

14 — storage has been allocated (after a call with IV(1) = 13 — see, for example, §4a below).

Error returns that preclude restarts:

15 — LIV too small.

16 — LV too small.

17 — restart attempted (§13) with problem dimensions changed.

18 — *d* has a negative component and IV(DTYPE) ≤ 0: see §4.

19–43 — V(IV(1)) is out of range.

44–62 — reserved.

63 — $f(x)$ cannot be computed at the initial *x*.

64 — bad parameters on an internal call (should not occur).

65 — the gradient could not be computed at *x*.

66 — bad input array — if this return is relevant, the associated PORT reference sheet will say so and explain what is good and bad.

67 — bad first parameter (`KIND` in §2) to IVSET.

68–69 — bugs encountered (should not occur).

70 — couldn't get initial *S* matrix by finite differences (regression routines only, and only when IV(INITS) is at least 3). [The *S* matrix is an approximation to part of the Hessian matrix, $\nabla^2 f$;

see [1] for details.]

71–79 — reserved.

80 — IV(1) was out of range (e.g. exceeded 14).

81 — bad problem dimensions (e.g. a nonpositive number of variables or, for regression routines, number of observations).

82 — inconsistent bounds in (1.1): $\underline{b}_i^x > \overline{b}_i^x$ for some i.

83 — inconsistent bounds in (1.2): $\underline{b}_i^c > \overline{b}_i^c$ for some i.

84 — some row of the constraint matrix, C in (1.2), is all zeros.

85 — inconsistent constraints: there is no x that satisfies both (1.1) and (1.2). (Routines that handle general linear constraints let you specify both (1.1) and (1.2).)

4. Scaling

A scale vector $d = (d_1, d_2, \cdots, d_p)$ is used both in the convergence tests and in computing trial values of x. Its choice can profoundly affect the performance of the optimization codes. By default, the regression routines choose d adaptively (since a reasonable choice is available in this case from the associated Jacobian matrix), but the general optimization routines require you to provide d as an input parameter. (The higher level regression routines allocate storage for d within V — see §4b and §4d.) d should be such that $|d_i \cdot x_i|$ are all ''comparable'' (e.g. are in comparable units), $1 \le i \le p$. Often you can get a reasonable choice of d by guessing upper bounds ξ_i on $|x_i^*|$ and setting $d_i := 1/\xi_i$.

Many problems are naturally well scaled in the sense that $d_i := 1$ for all i works well. You can have d set to all ones by setting V(DINIT) = V(38) to 1.0 and IV(DTYPE) = IV(16) to 0. (For general optimization, IV(DTYPE) is 0 by default, so it is unnecessary to change it in this case.)

Below it will be convenient to let D denote the diagonal matrix whose ith diagonal element is d_i, where d is the current scale vector:

$$D = \mathrm{diag}(d_1, d_2, \cdots, d_p). \tag{4.1}$$

§§4a–c below describe various IV and V components connected with scaling; §4d summarizes the relevant symbolic subscripts and default input values.

4a. Adaptive scaling for regression

Associated with regression problems is an $n \times p$ Jacobian matrix J (described further on the relevant PORT reference sheet). Let J_i denote the ith column of J, $J_i = (J_{1,i}, J_{2,i}, \cdots, J_{n,i})^T$. The adaptive choice of d_i uses the norm (1.3) of J_i to update d_i every time a new J is computed:

$$d_i := \max\{\mathrm{V(DFAC)} \cdot d_i,\ \|J_i\|\}; \tag{4.2}$$

$$\text{if } d_i < DTOL_i \text{ then } d_i := d_i^\circ \tag{4.3}$$

for $1 \le i \le p$. The $DTOL$ and d° arrays are stored in V and initialized as explained in §4d below; the factor V(DFAC) = V(41) that appears in (4.2) is 0.6 by default. This factor is included to keep d_i from shrinking too quickly. The $DTOL$ and d° arrays provide a ''floor'' on the d_i values — some problems have points where $\|J_i\|$ gets very small; when $\|J_i\|$ gets too small, it is often better to set d_i to a larger value, $d^\circ{}_i$, than the floor value $DTOL_i$.

Occasionally it may be useful to set individual components of $DTOL$ or d° to different values. To do this, first call IVSET (see §2), then set V(DTINIT) or V(D0INIT) to 0 and set IV(1) to 13. Next, call the optimization code: if it finds nothing wrong, it will return with IV(1) = 14, meaning that it has only allocated storage (within IV and V). Now determine from IV(DTOL) — see §4d — where the $DTOL$ and d° arrays are located and assign whatever values you like to them (making sure to assign *all* components). Finally, call the optimization code again: it will begin its algorithm. (The algorithm will not further change the $DTOL$ and d° arrays, but will use them in adaptively updating the scale vector d by (4.2) and (4.3).)

4b. Fixed scaling for regression.

For regression problems, you may specify fixed rather than adaptive scaling, in which case you supply your choice of values for the *d* vector by a procedure analogous to that in the previous paragraph. First call IVSET, then set V(DINIT) to –1.0, IV(DTYPE) to 0, and IV(1) to 13, then call the optimization code and make sure it has set IV(1) to 14 (i.e., has found nothing wrong). Now you can determine from IV(D) (see §4d) where *d* is stored and can assign the desired values to it; be sure to assign values to *all* components of *d*. Finally, call the optimization code again: it will begin its algorithm.

4c. Adaptive scaling for general optimization

An adaptive choice of *d* is available only to routines such as MNH and MNHB that are explicitly given the Hessian matrix $\nabla^2 f(x)$. The adaptive scaling update is similar to (4.2) and (4.3):

$$d_i := \begin{cases} \max\{\mathrm{V(DFAC)}\cdot d_i,\ |\nabla^2 f_{ii}(x)|^{1/2}\} & \text{if } |\nabla^2 f_{ii}(x)|^{1/2} \ge DTOL_i \\ \max\{\mathrm{V(DFAC)}\cdot d_i,\ d_i^\circ\} & \text{otherwise}^1 \end{cases}.$$

To turn this updating on, you must set IV(DTYPE) to 1 or 2 and must set V(DINIT) to 0.0.

4d. IV and V components that control scaling

The IV and V components below appear in alphabetical order.

`IV(D)` — IV(27) is the subscript of V at which the *d* (scaling) array starts [regression only].

`IV(DTOL)` — IV(59) is the subscript of V at which the *DTOL* array starts, and IV(DTOL) + *p* is the subscript for V at which the d° array starts. Both arrays are used in updating *d* — see §4a. (Recall that *p* is the number of *p*arameters, i.e., components in *x*.)

`IV(DTYPE)` — IV(16) tells whether *d* should be updated (when updating is possible — some optimization codes disallow it and ignore IV(DTYPE)).
0 means do not update *d*.
1 means update *d* every iteration.
2 means update *d* on the first iteration only. This occasionally works better than IV(DTYPE) = 0.
Default = 1 for regression, 0 for general optimization.

`V(D0INIT)` — V(40), if positive, is the value to which the d° array used in updating *d* is initialized — see §4a.
Default = 1.0.

`V(DFAC)` — V(41) is used in updating *d* — see §4a.
Default = 0.6.

`V(DINIT)` — V(38), if nonnegative, is the value to which the optimization routine initializes all components of *d* before it does any updating of *d*. If V(DINIT) < 0, then the optimization routine will not initialize *d*.
Default = 0.0 for regression, –1.0 for general optimization.

`V(DTINIT)` — V(39), if positive, is the value to which the optimization routine initializes all components of the *DTOL* array used in updating *d* — see §4a. If V(DTINIT) ≤ 0, then the optimization routine will not initialize the *DTOL* array.
Default = 10^{-6}.

1. The 1984 *Usage Summary* omitted the factor of V(DFAC) here.

5. Stopping tolerances

The same stopping tests are available in all PORT optimization routines covered by this usage summary. These stopping tests are designed so you can say how close x should be to a local minimizer $x*$ or how close $f(x)$ should be to $f(x*)$ before a favorable ''convergence'' return occurs. The default stopping tolerances are stringent enough that such favorable convergence returns are seldom misleading. This stringency may add little to the expense of minimizing f when $f(x)$ is evaluated accurately and when $x*$ is well defined. But if f is ''noisy'', e.g. if you must solve a differential equation, do some numerical quadrature, or perform an elaborate simulation to compute $f(x)$, then you will probably have to relax the default stopping tolerances. See §9 for more discussion of this matter.

In alphabetical order, the IV and V components controlling the stopping tests are:

`IV(MXFCAL)` — IV(17) is the maximum number of function evaluations allowed. If IV(MXFCAL) evaluations of $f(x)$ occur before another stopping test is satisfied, then you get a return with IV(1) = 9.
Default = 200.

`IV(MXITER)` — IV(18) is the maximum number of iterations allowed. (There is generally one gradient evaluation per iteration.) If IV(MXITER) iterations occur before another stopping test is satisfied, then you get a return with IV(1) = 10.
Default = 150.

`V(AFCTOL)` — V(31) is the *absolute function-convergence* tolerance. A return with IV(1) = 6 for absolute function convergence occurs if $|f(x)| < \text{V(AFCTOL)}$. This test is only of interest on problems where $f(x*) = 0$ is possible, such as fitting problems with artificial (exact) data. [The relative function convergence test described below fails when convergence to a $f(x*) = 0$ occurs, and the X-convergence test described below fails when convergence to $x* = \mathbf{0}$, i.e., $x_i* = 0$ for all i, occurs. People like to construct simple test examples having both $x* = \mathbf{0}$ and $f(x*) = 0$.]
Default = 10^{-20}.

`V(LMAXS)` — V(36) is used in the singular-convergence test described below with V(SCTOL).
Default = 1.0.

`V(RFCTOL)` — V(32) is the *relative function-convergence* tolerance. A return with IV(1) = 4 (or 5) occurs if the algorithm thinks $f(x) - f(x*) \le \text{V(RFCTOL)} \cdot |f(x)|$
Default = $\max\{10^{-10},\ \text{MACHEP}^{2/3}\}$.

`V(SCTOL)` — V(37) is the *singular-convergence* tolerance. A return with IV(1) = 7 occurs if a more favorable stopping test is not satisfied and if the algorithm thinks

$$f(x) - \min\{f(y): \|D(y-x)\| \le \text{V(LMAXS)}\} < \text{V(SCTOL)} \cdot |f(x)|,$$

where D is given by (4.1). When this test is satisfied, it appears that x has too many degrees of freedom — and you should ponder whether f was properly formulated.
Default = $\max\{10^{-10},\ \text{MACHEP}^{2/3}\}$.

`V(XCTOL)` — V(33) is the *X-convergence* tolerance. A return with IV(1) = 3 (or 5) occurs if the algorithm thinks the scaled distance from x to $x*$ is at most V(XCTOL). This scaled distance, $\rho(x, x*)$, is defined by

$$\rho(x, y) := \frac{\max\{d_i \cdot |x_i - y_i| : 1 \le i \le p\}}{\max\{d_j \cdot (|x_j| + |y_j|) : 1 \le j \le p\}}, \tag{5.1}$$

where d is the scale vector (§4).
Default = $\text{MACHEP}^{1/2}$.

You may change (5.1) to whatever you like by supplying your own function RLDST (DRLDST in double precision) to compute $\rho(x, y)$. Your RLDST should begin

```
REAL FUNCTION RLDST(P, D, X, Y)
INTEGER P
REAL D(P), X(P), Y(P)
```

`V(XFTOL)` — V(34) is the *false-convergence* tolerance. A return with IV(1) = 8 occurs if a more favorable stopping test is not satisfied and if a step of scaled length at most V(XFTOL) is tried but not accepted. ''Scaled length'' is in the sense of (5.1). Such a return generally means there is an error in computing $\nabla f(x)$, or the favorable convergence tolerances (V(RFCTOL), V(XCTOL), and perhaps V(AFCTOL)) are too tight for the accuracy to which $f(x)$ is computed (see §9), or ∇f (or f itself) is discontinuous near x. An error in computing $\nabla f(x)$ usually leads to false convergence after only a few iterations — often in the first.

Default = 100·MACHEP.

6. Printed output

The Fortran output unit for printing is IV(PRUNIT) = IV(21), the default for which is the standard output unit number (`I1MACH(2)`). All printing may be turned off by setting IV(PRUNIT) to 0.

6a. Print controls

Several IV components determine what printing is done. All such printing is done by default.

`IV(COVPRT)` — IV(14) [regression routines only] controls printing of a covariance matrix and regression diagnostic array:

0 means print neither.
1 means print just an estimated covariance matrix.
2 means print just the diagnostic array.
3 means print both.

Default = 3.

If IV(COVPRT) > 0, and if the Hessian approximation used in computing the covariance matrix or regression diagnostics is positive definite, then an upper bound on the reciprocal of the Euclidean condition number (i.e., an upper bound on the ratio of smallest to largest eigenvalue) of this Hessian approximation will also be printed. If this number is very small (say less than .01 or .001), then you should regard the computed covariance matrix with considerable skepticism. In this case, if $|$IV(COVREQ)$|$ is 1 or 2 (see §10) then you are probably ''close'' to singular convergence (§5). See §10 below for more discussion. The routines that do the printing controlled by IV(COVPRT) are described in the PORT reference sheet for the relevant iteration driver.

`IV(DRADPR)` — IV(101) [routines for general linear constraints only] controls printing of messages about constraints dropped and added:

1 means print which constraints are dropped and added.
0 means omit this printing.

Default = 1.

Routines allowing general linear constraints let you specify both simple bounds (1.1) and general linear constraints (1.2). In the printing controlled by IV(DRADPR), the ith simple lower bound constraint $x_i \ge \underline{b}_i^x$ is denoted i, and the ith simple upper bound constraint $x_i \le \overline{b}_i^x$ is denoted $-i$; similarly, the ith general lower bound constraint $\sum_{j=1}^{p} C_{i,j} x_j \ge \underline{b}_i^c$ is denoted iG, and the ith general upper bound constraint $\sum_{j=1}^{p} C_{i,j} x_j \le \overline{b}_i^c$ is denoted $-i$G. Thus 2 means the second simple lower bound constraint and –3G means the third general upper bound constraint.

Subroutine DRADP (DDRADP in double precision) does the printing controlled by IV(DRADPR): see the source code for details.

`IV(OUTLEV)` — IV(19) controls the printing of an iteration summary. If IV(OUTLEV) is nonzero, then an iteration summary is printed every |IV(OUTLEV)| iterations. If IV(OUTLEV) = 0, then no iteration summary is printed. For IV(OUTLEV) > 0, long summary lines are printed, and for IV(OUTLEV) < 0, short summary lines are printed. See §6b for details.

Default = 1.

Subroutine ITSUM (DITSUM in double precision) does the printing controlled by IV(OUTLEV): see §6c.

`IV(PARPRT)` — IV(20) controls printing of nondefault V (and a few nondefault IV) input components. IV(PARPRT) = 1 causes them to be printed, and IV(PARPRT) = 0 suppresses this printing.

Default = 1.

Subroutine PARCK (DPARCK in double precision) does the printing controlled by IV(PARPRT): see §6c.

`IV(PRUNIT)` — IV(21) is the Fortran unit number on which all printing (other than error messages from the top-level PORT versions of the optimization codes) is done. You can turn all this printing off at once by setting IV(PRUNIT) to 0.

Default = *standard output unit* = `I1MACH(2)`.

`IV(SOLPRT)` — IV(22) controls printing of the returned x, scale vector d, and (except for routines with general linear constraints) the gradient $\nabla f(x)$. IV(SOLPRT) = 1 means provide this printing, and IV(SOLPRT) = 0 means omit it.

Default = 1.

For general linear constraints, IV(SOLPRT) also controls printing of Lagrange multipliers and a list of the constraints (1.1) and (1.2) that are active or redundant at x^{final} (the returned x). (''Redundant'' constraints are those whose normals are linearly dependent on the normals of the other active constraints. x^{final} would satisfy the same stopping test with these constraints removed. ''Active'' constraints are those that the algorithm regards as equality constraints at x^{final}. The constraints are denoted as explained above with IV(DRADPR); the signs of the multipliers are explained with IV(AM) in §14.) IV(SOLPRT) = 1 means provide all this printing, and IV(SOLPRT) = 0 means omit all of it. Finer control is also possible: use the following procedure to request some but possibly not all of of this printing.

- Set IV(SOLPRT) to 1.
- If you want to have x^{final} and the returned scale vector d printed, set IV(SOLPRT) to IV(SOLPRT) + 1.
- If you want to have the indices of the active or redundant constraints printed, set IV(SOLPRT) to IV(SOLPRT) + 2.
- If you want to have the Lagrange multipliers printed, set IV(SOLPRT) to IV(SOLPRT) + 4.

Thus, for example, IV(SOLPRT) = 8 has the same effect as IV(SOLPRT) = 1, and IV(SOLPRT) = 4 causes printing of the Lagrange multipliers to be omitted.

Unless specified otherwise in the relevant PORT reference sheet, subroutine ITSUM (DITSUM in double precision) does the printing controlled by IV(SOLPRT): see §6c.

`IV(STATPR)` — IV(23) controls printing of a one-line convergence or error return message and printing of summary statistics once a stopping test has been satisfied. IV(STATPR) = –1 suppresses all this printing; IV(STATPR) = 0 suppresses just the summary statistics; IV(STATPR) = $-n$ causes printing of return messages only for IV(1) > n. The summary statistics include the final $f(x)$, the RELDX, PRELDF, and NPRELDF values from the iteration summary (§6b) for the final iteration (even if printing of the iteration summary is turned off), the number of function evaluations performed (excluding those for finite differences and for computing a covariance matrix or regression diagnostics), the number of gradient evaluations performed (or, for finite-difference gradients, the number of extra function evaluations done to compute gradients — again excluding evaluations for computing a covariance matrix or regression diagnostics), and the number of extra function and gradient evaluations (if any) done for computing a covariance matrix or regression diagnostics.

Default = 1.

Subroutine ITSUM (DITSUM in double precision) does the printing controlled by IV(STATPR): see §6c.

`IV(X0PRT)` — IV(24) controls printing of the initial *x*. IV(X0PRT) = 1 means provide this printing, and IV(X0PRT) = 0 means omit it.

Default = 1.

Subroutine ITSUM (DITSUM in double precision) does the printing controlled by IV(X0PRT): see §6c.

6b. Iteration summary

Printing of an iteration summary is controlled by IV(OUTLEV) — see §6a above. Either a long or a short summary is possible; short summary lines are long summary lines with the last two columns omitted. Columns in the long iteration summary include:

`IT` — the iteration number for this summary line.

`NF` — the number of function evaluations (computations of $f(x)$) so far computed, excluding those for finite differences. The number of additional function evaluations for finite differences is reported only in the summary statistics controlled by IV(STATPR) — see §6a above and IV(NGCALL) in §14.

`F` — the current value of $f(x)$. For nonlinear least squares, this is *half* of the residual sum of squares at *x*.

`RELDF` — $[f(x^{prev}) - f(x)] \,/\, \max\{|f(x^{prev})|, |f(x)|\}$, the relative function reduction achieved in the current iteration (where x^{prev} is the *x* value from the end of the previous iteration).

`PRELDF` — the value of RELDF that the algorithm predicted.

`RELDX` — $\rho(x^{prev}, x)$, the scaled length of the step taken in this iteration, where ρ is given by (5.1).

`MODEL` — [regression routines only] the model or sequence of models used in the iteration: ‘‘G’’ means Gauss-Newton model, ‘‘S’’ means augmented model. See [1] for details on these models.

`STPPAR` — step-length (e.g. Levenberg-Marquardt) parameter for the step just taken: 0 means a full Newton step, positive means a damped step, negative means a damped step in which the special case described in [3] was detected.

`D*STEP` — $\| D(x - x^{prev}) \|$ where D is given by (4.1).

`NPRELDF` — the value of RELDF predicted for a full Newton step (for NPRELDF > 0 or NPRELDF = STPPAR = 0). This is the quantity used in the relative function-convergence test described in §5. NPRELDF < 0 means –NPRELDF is the value against which V(SCTOL) is compared in the singular-convergence test (§5). When NPRELDF > 0 and STPPAR ≠ 0 (i.e., the algorithm does not take a full Newton step), PRELDF will generally be less than NPRELDF, since PRELDF corresponds to the step actually taken.

Subroutine ITSUM (DITSUM in double precision; see §6c) prints the iteration summary.

6c. Print routine calling sequences

```
SUBROUTINE ITSUM(D, G, IV, LIV, LV, P, V, X)
INTEGER LIV, LV, P
INTEGER IV(LIV)
REAL D(P), G(P), V(LV), X(P)

SUBROUTINE PARCK(KIND, D, IV, LIV, LV, P, V)
INTEGER KIND, LIV, LV, P
INTEGER IV(LIV)
REAL D(P), V(LV)
```

In addition to the ubiquitous IV and V (and their lengths LIV and LV), parameters to ITSUM include the current scale vector D (see §4), the problem dimension P (see §1a), the current gradient vector G = $\nabla f(x)$, and the current iterate X = x.

Parameter KIND to PARCK comes from Table 1 of §2; the other parameters are the same as for ITSUM (omitting G and X). In addition to optionally printing nondefault IV and V input components, PARCK initializes IV(LASTIV), IV(LASTV), IV(NEXTIV), and IV(NEXTV) (all described in §14) and checks the validity of various inputs.

7. Initial step bound

The algorithms maintain an estimate of the diameter of a region about the current x in which they can predict the behavior of f reasonably well. This region has the form $\{y\colon \|D(y - x)\| \le \delta\}$, where D is given by (4.1). The initial δ (the one used at the start of the very first iteration) is given by V(LMAX0) = V(35), whose default value is 1.0.

The choice of V(LMAX0) can profoundly affect the performance of the algorithms — different values sometimes lead to finding different local minimizers $x*$. Too small or too large a value of V(LMAX0) causes the algorithm to spend several function evaluations in the first iteration increasing or decreasing δ. If the iteration summary line (see §6b) for the first iteration (the line with 1 in the IT column) shows more than one function evaluation performed (the number in the NF column) then you would have saved some function evaluations had V(LMAX0) had the value in the D*STEP column of the same summary line. If you will be solving several similar problems, you may wish to examine the iteration summary for the first problem and then choose an appropriate nondefault value for V(LMAX0) on the subsequent problems.

8. Finite differences

The following V components affect various finite-difference computations. For noisy functions (§9), it may be necessary to relax the relevant component(s):

`V(DELTA0)` — V(44) [regression routines only] helps choose the step sizes for computing a finite-difference Hessian approximation from gradient differences (i.e., when IV(COVREQ) = 1 or 2) for use in computing a covariance matrix, regression diagnostics, or initial S matrix (§17). For differences involving x_i, step size

$$\mathrm{V(DELTA0)} \cdot \max\{|x_i|,\ d_i^{-1}\} \cdot \mathrm{sign}(x_i)$$

is first tried.

Default = MACHEP $^{1/2}$.

`V(DLTFDC)` — V(42) [regression routines only] helps choose the step sizes for computing a finite-difference Hessian approximation from function differences (i.e., when IV(COVREQ) = –1 or –2) for use in computing a covariance matrix, regression diagnostics, or initial S matrix (§17). For differences involving x_i, step size

$$\mathrm{V(DLTFDC)} \cdot \max\{|x_i|,\ d_i^{-1}\}$$

is first tried.

Default = MACHEP $^{1/3}$.

`V(DLTFDJ)` — V(43) [regression routines only] helps choose the step sizes for computing finite-difference Jacobian approximations. For differences involving x_i, step size

$$\mathrm{V(DLTFDJ)} \cdot \max\{|x_i|,\ d_i^{-1}\}$$

is first tried.

Default = MACHEP $^{1/2}$.

`V(ETA0)` — V(42) [general optimization only] helps choose the step sizes for computing finite-difference gradient approximations. You should set V(ETA0) to your best guess at a bound on the relative error in the computed values of $f(x)$: V(ETA0) should be such that if $\tilde{f}(x)$ is the value computed, then the true value $f(x)$ satisfies $f(x) = \tilde{f}(x) \cdot (1 + \varepsilon)$, where $|\varepsilon| \le V(ETA0)$. The scheme used is a slight modification of one proposed by Stewart [6]. See [4] for details.

Default = $10^3 \cdot$MACHEP.

The phrase ‘‘first tried’’ deserves explanation. Some of the finite-difference routines multiply the step size by .5 or –.5 and try again if the first step they try is rejected (§11).

9. Noisy functions

Sometimes evaluating $f(x)$ involves an extensive computation, such as performing a simulation or adaptive numerical quadrature or integrating an ordinary or partial differential equation. In such cases the value computed for $f(x)$, say $\tilde{f}(x)$, may involve substantial error (in the eyes of the optimization algorithm). To eliminate some ‘‘false convergence’’ messages and useless function evaluations, it is necessary to increase the stopping tolerances and, when finite-difference derivative approximations are used, to increase the step-sizes used in estimating derivatives.

Intelligently choosing these tolerances requires you to have a good estimate η of the maximum relative error in $\tilde{f}(x)$, i.e., of η such that

$$|\tilde{f}(x) - f(x)| \le \eta |\tilde{f}(x)|. \tag{9.1}$$

Often η is an input to the procedure that computes $\tilde{f}(x)$. At other times estimating η may be more difficult; see §8.5 of [5] for more discussion.

Once you have an approximate η, try setting the convergence tolerances V(RFCTOL) and V(SCTOL) as follows:

$$\mathrm{V(RFCTOL)} = \mathrm{V(32)} := \eta \ \textit{or}\ 10\eta,$$

$$\mathrm{V(SCTOL)} = \mathrm{V(37)} := \eta.$$

If you are requesting finite-difference derivative approximations (§8), try using $\eta^{1/2}$ for V(DELTA0) and V(DLTFDJ), $\eta^{1/3}$ for V(DLTFDC), or η for V(ETA0), as appropriate.

When you can specify η, perhaps as an accuracy tolerance to an integration routine, you will generally find that the smaller you make η, the more expensive $\tilde{f}(x)$ is to compute. In this case you can reduce the cost of computing $\tilde{f}(x)$ by requesting only enough accuracy to make the optimization routine happy. You can see from V(PREDUC) = V(7) what the algorithm predicts $\mathrm{V(F0)} - f(x)$ to be, where V(F0) = V(13) is the value $f(x)$ had at the start of the iteration, and you will often find it satisfactory to specify $\eta = 10^{-2} \cdot \mathrm{V(PREDUC)} / \mathrm{V(F0)}$ or perhaps even $\eta = 10^{-1} \cdot \mathrm{V(PREDUC)} / \mathrm{V(F0)}$. When using finite-difference derivative approximations, you should probably not tinker with η in this way. And for regression routines, you should first check IV(MODE) = IV(35): if IV(MODE) > 0, then a finite-difference Hessian computation is under way, and you should provide accuracy consistent with V(DELTA0) [or V(DLTFDC)] — see §8.

Gaining access to IV and V (for tinkering with η as in the previous paragraph) deserves some discussion. If you call a reverse-communication driver (§1c), then you have immediate access to IV and V whenever the driver requests a new $f(x)$ value. If you call a forward-communication optimization routine, then you must provide a subroutine, say CALCF, to compute either $f(x)$ itself (for general optimization) or the information needed to compute $f(x)$ (for regression). In either case the parameters to both the optimization routine and CALCF include ‘‘user’’ integer and floating-point arrays UI and UR; you could pass IV for UI and V for UR, or you could put IV and V into a common block that CALCF knows about.

10. Covariance, regression diagnostics, and confidence intervals

CAVEAT: *An estimated covariance matrix and the confidence intervals derived from it may be worthless if the assumptions behind the covariance computation are invalid. See the discussion of confidence intervals below.*

Regression routines may offer an estimated covariance matrix Ξ and regression diagnostic vector *RD* at the computed solution x^{final}, but only for favorable returns ($3 \le \mathrm{IV}(1) \le 6$ — see §3). The *i*th component of *RD* is an estimate of the square-root of twice the relative[2] (or, if $f(x^{final}) = 0$, absolute) change that would occur in $f(x^{final})$ if the *i*th observation were deleted; you may wish to take a closer look at the observations corresponding to large components of *RD*. (Because of the square-root used in defining the *RD* values, if deleting component *i* would cause α times the estimated change in x^{final} as deleting component *j*, and if both changes were in the same direction, then $RD(i) = |\alpha| RD(j)$.) The following IV components control whether and how Ξ and *RD* are computed:

`IV(COVREQ)` — IV(15) tells what kind of Hessian approximation H should be used in the computations:

0, 1 and 2 request a finite-difference H computed from gradient differences.
–1 and –2 request a finite-difference H computed from function differences.
3 and –3 request $H = J^T J$, where J is the Jacobian matrix at x^{final}.

For nonlinear least-squares, $|\mathrm{IV(COVREQ)}| \le 1$ requests a covariance matrix of the form

$$\Xi = \sigma^2 H^{-1} J^T J H^{-1},$$

$|\mathrm{IV(COVREQ)}| = 2$ requests a covariance matrix of the form

$$\Xi = \sigma^2 H^{-1},$$

and $|\mathrm{IV(COVREQ)}| \ge 3$ requests a covariance matrix of the form

$$\Xi = \sigma^2 (J^T J)^{-1},$$

where σ^2 is the residual sum of squares divided by $\max\{1,\ n - p\}$, n being the number of observations. $|\mathrm{IV(COVREQ)}| = 1$ is perhaps the most defensible choice, but the others have their proponents.

Default = 1.

`IV(RDREQ)` — IV(57) tells whether to compute a covariance matrix or regression diagnostic array:

0 means compute neither;
1 means compute just a covariance matrix;
2 means compute just the regression diagnostic array;
3 means compute both.

Default = 3.

Printing of Ξ and *RD* is described in §6a. You can obtain their numerical values by looking at the appropriate IV and V components:

2. The 1984 *Usage Summary* omitted ‘‘square-root of twice’’ here. To remove dependence on the scale of *f*, ‘‘relative’’ was added in October, 1990.

`IV(COVMAT)` — IV(26), if positive, is the starting subscript in V for the lower triangle of Ξ, which is stored compactly by rows: $\Xi_{1,1}$, $\Xi_{2,1}$, $\Xi_{2,2}$, $\Xi_{3,1}$, $\Xi_{3,2}$, $\Xi_{3,3}$ $\cdots$. Other possible values for IV(COVMAT) include:

0 for no covariance matrix computation attempted;
–1 for an indefinite H [regard this as similar to singular convergence (IV(1) = 7 — see §3)].
–2 for too many x values rejected (see §11) during finite-difference computation of H.

`IV(REGD)` — IV(67), if positive, is the starting subscript in V for the regression diagnostic array *RD*. IV(REGD) ≤ 0 means the same as IV(COVMAT) having this value.

CONFIDENCE INTERVALS for nonlinear least squares: You may wish to know confidence intervals for the components x_i of the returned solution. You have several options, but explaining them requires some notation. Suppose you have n observations and a model ϕ that attempts to explain them — $\phi = (\phi_1(x),\ \phi_2(x),\ \cdots,\ \phi_n(x))^T$. You decide to estimate the model parameters x by least squares: $f(x) := \frac{1}{2} \sum_{j=1}^{n} (\phi_j(x) - y_j)^2$. The diagonal entries of the covariance matrix computed with $|\mathrm{IV(COVREQ)}| = 1$ are estimates of the variances (i.e., standard deviations squared) of the x_i. Two assumptions underlie these estimates: (1) that the observations y_j are subject to independent errors whose variances are well estimated by $1 / (n - p)$ times the residual sum of squares (i.e., $2 \cdot f(x^{final}) / (n - p)$), and (2) that approximating *f* by a second-order Taylor expansion does not introduce ''too much'' error into the estimated covariance matrix. The first assumption is a standard one, but it may not apply to your problem — if you have a better estimate σ^2_{better} for the variances of the y_j, then you should scale the returned covariance matrix by $\frac{1}{2} \cdot \sigma^2_{better} \cdot (n - p) / \mathrm{V(F)}$, where V(F) = V(10) = $f(x^{final})$. The second assumption generally founders when the variances of the y_j are too large, where ''large'' depends on how nonlinear ϕ is; you might have to resort to Monte-Carlo techniques to compute a realistic covariance matrix and confidence intervals. (If you do, see the end of §17.) At any rate, in posing the problem to the regression routine, it is important that you scale the components of ϕ and y (correspondingly) so the variances of the errors to which they are subject are all about the same.

The following code sets STDDEV(i) to an estimate of the standard deviation of x_i in the case where you know σ^2_{better}:

```
      T = 0.5*σ²better *MAX0(1,N-P)/V(F)
      II = IV(COVMAT) - 1
      DO 10 I = 1, P
         II = II + I
         STDDEV(I) = SQRT(T*V(II))
 10      CONTINUE
```

If assumption (1) is valid, omit the first line as well as ''`T*`'' in the penultimate line.

After making further assumptions about the errors, you can relate STDDEV(i) to a confidence interval for x_i. For example, if the errors in the y_j are normally distributed with zero mean, then [x_i^{final} – 1.96·STDDEV(i), x_i^{final} + 1.96·STDDEV(i)] is a 95% confidence interval for x_i. (Change 1.96 to 1.645 to get a 90% confidence interval and to 2.576 to get a 99% confidence interval.)

11. Identifying (or rejecting) *x*

When you compute $\nabla f(x)$ analytically, you must often use intermediate quantities that are also needed for computing $f(x)$. You may find it convenient to save such quantities for use in computing $\nabla f(x)$. But there is a complication: the optimization routine may first ask for $f(x^a)$, then $f(x^b)$, then $\nabla f(x^a)$, i.e., the x at which ∇f is evaluated may not be the one at which *f* was most recently evaluated. It usually suffices to save two sets of intermediate quantities, corresponding to the two x values at which *f* was most recently evaluated. You can use the invocation count for *f* to identify these sets of intermediate quantities. If you use a forward-communication optimization routine, one to which you supply, say, subroutines CALCF and CALCG for computing $f(x)$ and $\nabla f(x)$, then the calling sequence for these subroutines includes an integer parameter NF that you can use to identify *x*. For CALCF, NF is the invocation count for CALCF (i.e., the number of times CALCF has been called); for CALCG, NF is the value that was supplied to CALCF when CALCF was called with the *x* now being passed to CALCG. If you are calling a reverse-communication routine (§1c), then the NF value that would be passed to CALCF is in IV(NFCALL) = IV(6), and the NF value that would be passed to CALCG is in IV(NFGCAL) = IV(7). (Do not change IV(NFCALL) or IV(NFGCAL); subroutines CALCF and CALCG should not change NF unless they wish to reject *x*, as described in the next paragraph.)

Rejecting *x*: Sometimes the optimization routine will attempt too large a step or will otherwise request that *f* be evaluated outside its effective domain. You can tell the routine to back off and try a shorter step as follows: if you are using a forward-communication routine (to which you pass CALCF), then have CALCF set NF to 0 and return; if you are calling a reverse-communication routine, set IV(TOOBIG) = IV(2) to 1. (CALCG can also reject *x* by setting NF to 0, or you can reject *x* by setting IV(TOOBIG) to 1 when a reverse-communication iteration driver asks you to compute $\nabla f(x)$, but then you will get an error return: the routines assume something serious is wrong if they encounter an *x* where $f(x)$ can be evaluated but $\nabla f(x)$ cannot.)

12. STOPX

If you use the PORT optimization routines in an interactive environment, then you can arrange for them to respond to the ‘‘BREAK’’ key — to check before each evaluation of $f(x)$ whether ‘‘BREAK’’ has been pressed and to return in a way that allows restarts (§13) if so. To do this, you must supply a logical function STOPX that returns .TRUE. exactly when ‘‘BREAK’’ has been pressed since the last time STOPX was called. Your STOPX (which will likely be written in a language other than Fortran) should behave as though it began

```
LOGICAL FUNCTION STOPX(DUMMY)
INTEGER DUMMY
```

STOPX should ignore its parameter DUMMY (which is required by the syntax of Fortran 66). When STOPX returns .TRUE., you get a return with IV(1) = 11.

13. Restarting

If you get a return with IV(1) < 12 (from either a forward- or reverse-communication optimization routine — see §1c and §3), then you can resume the algorithm where it left off. Just invoke the routine again, usually after changing the IV or V input components responsible for the return you got. You can even write the IV and V arrays, *x*, and other parameters (as appropriate) on an auxiliary storage device, read them in later (perhaps in another session), and then resume the algorithm. This is sometimes useful for checkpointing or debugging.

14. Output IV components

You can probably skip this section. It describes IV components (listed alphabetically) that are given values by the relevant PORT optimization routines.

`IV(A)` — IV(98) [general linear constraints only] is the starting subscript in IV for a permutation array, $a = (a_1, a_2, \cdots, a_{m+p})$, where *m* is the number of general constraints (1.2). Together with IV(ME), IV(ME1), IV(MC), and IV(PC) (all described below), *a* tells how the algorithm regards the various constraints at x^{final}, the returned *x*: for $1 \le i \le$ IV(ME1), constraint a_i is a redundant equality constraint (see IV(SOLPRT) in §6a); for IV(ME1) $< i \le$ IV(ME1) + IV(ME), constraint a_i is a (nonredundant) equality constraint; and for IV(ME1) + IV(ME) $< i \le$ IV(ME1) + IV(ME) + IV(MC), constraint a_i is an active inequality constraint. (If IV(MC) + IV(PC) > *p*, then the last *p* – [IV(MC) + IV(PC)] such inequality constraints are redundant at x^{final}.)

The numbering within *a* of the constraints deserves an explanation. Routines that handle general linear constraints allow both simple bounds (1.1) and general linear constraints (1.2). The constraints are numbered as follows: for $1 \le i \le p$, the simple lower-bound constraint $\underline{b}_i^x \le x_i$ has index i and the simple upper-bound constraint $x_i \le \overline{b}_i^x$ has index $-i$; for $1 \le i \le m$, the general lower-bound constraint $\underline{b}_i^c \le \sum_{j=0}^{p} C_{ij} x_j$ has index $i + p$ and the general upper-bound constraint $\sum_{j=0}^{p} C_{ij} x_j \le \overline{b}_i^c$ has index $-(i + p)$.

`IV(AI)` — IV(91) [general linear constraints only] is the starting subscript in IV for an array $\tilde{a}$ of indices of the Lagrange multipliers corresponding to x^{final}. (The entries in this array also appear in the a array described above with IV(A), but in a different order — here they are sorted on their absolute values. For more on the multipliers themselves, see IV(AM) below.) There are p – IV(PC) such multipliers (see IV(PC) below).

`IV(AM)` — IV(95) [general linear constraints only] is the starting subscript in V for the Lagrange multiplier array λ mentioned with IV(AI) above. IV(SOLPRT) (see §6a) controls the printing of this array. The multipliers λ_i are such that if C in (1.2) is expanded to include (1.1) in the first p rows and C_i denotes the ith row of this expanded C, then

$$\sum_{i=1}^{p-\mathrm{IV(PC)}} \lambda_i \ \mathrm{sign}(\tilde{a}_i) \ C_{|\tilde{a}_i|} = \nabla f(x^{final}),$$

where $\tilde{a}$ is the index array described with IV(AI) above.

`IV(COVMAT)` — see §10.

`IV(D)` — IV(27) [regression routines only] is the starting subscript in V for the scale vector d (§4).

`IV(G)` — IV(28) [except for some reverse-communication iteration drivers] is the starting subscript in V for the gradient vector $\nabla f(x^{final})$.

`IV(LASTIV)` — IV(44) is the minimum acceptable value for LIV, the length of the IV array.

`IV(LASTV)` — IV(45) is the minimum acceptable value for LV, the length of the V array.

`IV(MC)` — IV(83) [general linear constraints only] is the number of inequality constraints (1.1) or (1.2) active at x^{final}, excluding equality constraints (see IV(ME) and IV(ME1) below and IV(A) above).

`IV(ME)` — IV(86) [general linear constraints only] is the number of nonredundant equality constraints — linearly independent constraints $\underline{b}_i^x \le x_i \le \overline{b}_i^x$ or $\underline{b}_i^c \le \sum_{j=0}^{p} C_{ij} x_j \le \overline{b}_i^c$ from (1.1) and (1.2) having $\underline{b}_i^x = \overline{b}_i^x$ or $\underline{b}_i^c = \overline{b}_i^c$. See also IV(A) above.

`IV(ME1)` — IV(87) [general linear constraints only] is the number of equality constraints being ignored because they are linearly dependent on the nonredundant equality constraints (see IV(ME) above). The total number of equality constraints is thus IV(ME) + IV(ME1). See also IV(A) above.

`IV(NEXTIV)` — IV(46) is the subscript of the next free component in IV. For most routines it should have the value IV(LASTIV) + 1, but some routines give it a smaller value (if they release scratch space used only when they start up).

`IV(NEXTV)` — IV(47) is the subscript of the next free component in V. Analogously to IV(NEXTIV), it usually has the value IV(LASTV) + 1.

`IV(NFCALL)` — IV(6) is the number of function evaluations (evaluations of $f(x)$) performed, including evaluations for computing a covariance matrix or regression diagnostics but excluding extra evaluations for computing finite-difference derivative approximations (see

IV(NGCALL) below).

`IV(NFCOV)` — IV(52) [regression only] is the number of function evaluations performed just for computing a covariance matrix or regression diagnostics, excluding extra evaluations for computing finite-difference derivative approximations (see IV(NGCALL) below). The number of function evaluations reported in the summary statistics controlled by IV(STATPR) (see §6a) is IV(NFCALL) – IV(NFCOV).

`IV(NGCALL)` — IV(30) is either the total number of gradient evaluations performed (when you provide analytic derivatives) or the number of additional function evaluations required to compute finite-difference derivative approximations. In the latter case, the total number of function evaluations is thus IV(NFCALL) + IV(NGCALL).

`IV(NGCOV)` — IV(53) is the number of additional gradient evaluations performed just for computing a covariance matrix or regression diagnostics. The number of gradient evaluations reported in the summary statistics controlled by IV(STATPR) (see §6a) is IV(NGCALL) – IV(NGCOV).

`IV(NITER)` — IV(31) is the number of iterations performed. The number of gradient evaluations (aside from those used in covariance or regression diagnostic computations) is usually IV(NITER) + 1 or IV(NITER), depending whether or not the final iteration produced an acceptable new iterate.

`IV(PC)` — IV(90) [general linear constraints only] is the dimension of the free variable space at x^{final}; see IV(A) above.

`IV(SUSED)` — IV(64) [regression only] describes the sequence of models for f considered in the last iteration (see [1]); IV(SUSED) determines what gets printed in the MODEL column of the iteration summary (§6b): 1 means ‘‘G’’, 2 means ‘‘S’’, 3 means ‘‘G–S’’ (i.e., the algorithm first tried the ‘‘G’’ model, then switched to the ‘‘S’’ model), 4 means ‘‘S–G’’, 5 means ‘‘G–S–G’’ (i.e., the algorithm first tried the ‘‘G’’ model, then the ‘‘S’’ model, then returned to the ‘‘G’’ model), and 6 means ‘‘S–G–S’’.

15. Output V components

This is another section most people can ignore. It describes (alphabetically) V components to which various PORT optimization routines supply values.

`V(DGNORM)` — V(1) = $\| D^{-1} \nabla f(x^{final}) \|$ where D is given by (4.1).

`V(DSTNRM)` — V(2) = $\| D(x - x^{prev}) \|$ the scaled Euclidean length of the last step taken (or, if the final iteration did not change x, of the last step attempted).

`V(F)` — V(10) is the current function value, $f(x^{final})$. For nonlinear least squares, this is *half* the current residual sum of squares.

`V(F0)` — V(13) is the function value of $f(x)$ at the start of the last iteration.

`V(NREDUC)` — V(6), if positive (or zero with V(STPPAR) = 0), is the maximum reduction in f that the algorithm thinks is yet possible. V(NREDUC) = 0 with V(STPPAR) > 0 means the current Hessian (or, for general linear constraints, projected Hessian — projected onto the free-variable space) is not positive definite. If V(NREDUC) < 0, then – V(PREDUC) ⁄ V(F0) is the quantity against which V(SCTOL) is compared in the singular-convergence test — see §5. The quantity NPRELDF described in §6b is V(NREDUC) ⁄ max{V(F), V(F0)}.

`V(PREDUC)` — V(7) is the function reduction predicted for the last step taken or attempted (the step corresponding to V(DSTNRM)). The quantity PRELDF described in §6b is V(PREDUC) ⁄ max{V(F), V(F0)}.

`V(RADIUS)` — V(8) is the current trust-region radius δ, described in §7.

`V(RCOND)` — V(53) [regression only — and only when a covariance matrix or regression diagnostic array is requested] is the reciprocal of the square-root of a lower bound on the Euclidean condition number of the final Hessian $\nabla^2 f(x^{final})$. Printing of V(RCOND)^2 is controlled by IV(COVPRT) — see §6a.

`V(RELDX)` — V(17) is the scaled length, defined by (5.1), of the last step taken or attempted (the step corresponding to V(DSTNRM)).

`V(STPPAR)` — V(5) is the step-length parameter described in §6b.

16. Other V components

There are a few obscure input V components with which you should seldom have to tinker. They are described in §3.15 of [2]. V(COSMIN), V(FUZZ), and V(RLIMIT) pertain only to regression, and their subscript values have changed: now V(COSMIN) = V(47), V(FUZZ) = V(45), and V(RLIMIT) = V(46). The other input V components described in §3.15 of [2] retain their old subscript values and apply to all the optimization routines covered by this usage summary.

17. Initial *S* matrix

This section applies to regression routines only. These routines use a ‘‘secant update’’ to obtain an approximation S to part of $\nabla^2 f$ — see [1] for details in the case of nonlinear least squares. By default the initial S matrix is set to all zeros. Occasionally it is useful to initialize S to a finite-difference estimate of the thing it approximates. (This is useful, for instance, if you want to start at a point where the Jacobian matrix vanishes but $\nabla^2 f$ is nonzero — and there exist people who want to do this!) You can have this done by setting IV(INITS) = IV(25) to 3 or 4; 3 means to use differences of function values to estimate the initial S, and 4 means to use differences of gradients[4]. You can also supply your own initial S matrix. The procedure for doing so is similar to the one described in §4a. First call IVSET (DIVSET for double precision — see §2), then set IV(INITS) to 2 (or to 1 if you want the Gauss-Newton model — the one that ignores S — to be tried first) and IV(1) to 13. Next, call the appropriate optimization routine and make sure it has set IV(1) to 14 (i.e., has found nothing wrong). Now IV(S) = IV(62) is the starting subscript in V for the lower triangle of S. Store your initial S there (compactly by rows: $S_{1,1}, S_{2,1}, S_{2,2}, S_{3,1}, S_{3,2}, S_{3,3} \cdots$). Finally, call the optimization routine again: it will begin its algorithm.

If you do a Monte-Carlo covariance matrix computation (by repeatedly choosing pseudorandom errors, adding them to your observations, and calling the regression routine again), then it is reasonable to have the starting guess for the second and subsequent calls on the regression routine be the solution from the previous call. In this case it is reasonable to set IV(INITS) to IV(MODEL) = IV(5), so that you start with the S matrix and model preference (i.e., initial decision whether to use S in computing the next step — see [1]) from the previous run.

3. Mention of the possibility of setting IV(INITS) to 4 was omitted from the 1984 *Usage Summary*.

18. Numerical values for symbolic subscripts

The following symbolic subscripts for IV and V are discussed in the indicated sections:

IV symbolic subscript values, sorted alphabetically

Symbol	*Value*	*Sections*	*Symbol*	*Value*	*Sections*
A	98	14	NEXTIV	46	6c, 14
AI	91	14	NEXTV	47	6c, 14
AM	95	6a, 14	NFCALL	6	11, 14
COVMAT	26	10, 14	NFCOV	52	14
COVPRT	14	6a, 15	NFGCAL	7	11
COVREQ	15	6a, 8, 10	NGCALL	30	6b, 14
D	27	4d, 14	NGCOV	53	14
DRADPR	101	6a	NITER	31	14
DTOL	59	4d	OUTLEV	19	6a
DTYPE	16	3, 4c	PARPRT	20	6a
G	28	14	PC	90	14
INITS	25	3, 17	PRUNIT	21	2, 6a
LASTIV	44	2, 6c, 14	RDREQ	57	10
LASTV	45	2, 6c, 14	REGD	67	10
MC	83	14	S	62	17
ME	86	14	SOLPRT	22	6a, 14
ME1	87	14	STATPR	23	6b, 14
MODE	35	9	SUSED	64	14
MODEL	5	17	TOOBIG	2	11
MXFCAL	17	3, 5	X0PRT	24	6a
MXITER	18	1, 3, 5			

V symbolic subscript values, sorted alphabetically

Symbol	*Value*	*Sections*	*Symbol*	*Value*	*Sections*
AFCTOL	31	3, 5	FUZZ	45	16
COSMIN	47	16	LMAX0	35	7
D0INIT	40	4a	LMAXS	36	5
DELTA0	44	8, 9	NREDUC	6	15
DFAC	41	4a	PREDUC	7	9, 15
DGNORM	1	15	RADIUS	8	15
DINIT	38	4b	RCOND	53	15
DLTFDC	42	8, 9	RELDX	17	15
DLTFDJ	43	8, 9	RFCTOL	32	3, 5, 9
DSTNRM	2	15	RLIMIT	46	16
DTINIT	39	4d	SCTOL	37	5, 6b, 9, 15
ETA0	42	8, 9	STPPAR	5	15
F	10	10, 15	XCTOL	33	3, 5
F0	13	9, 15	XFTOL	34	5

IV symbolic subscript values, sorted numerically					
Value	*Symbol*	*Sections*	*Value*	*Symbol*	*Sections*
2	`TOOBIG`	11	35	`MODE`	9
5	`MODEL`	17	44	`LASTIV`	2, 6c, 14
6	`NFCALL`	11, 14	45	`LASTV`	2, 6c, 14
7	`NFGCAL`	11	46	`NEXTIV`	6c, 14
14	`COVPRT`	6a, 15	47	`NEXTV`	6c, 14
15	`COVREQ`	6a, 8, 10	52	`NFCOV`	14
16	`DTYPE`	3, 4c	53	`NGCOV`	14
17	`MXFCAL`	3, 5	57	`RDREQ`	10
18	`MXITER`	1, 3, 5	59	`DTOL`	4d
19	`OUTLEV`	6a	62	`S`	17
20	`PARPRT`	6a	64	`SUSED`	14
21	`PRUNIT`	2, 6a	67	`REGD`	10
22	`SOLPRT`	6a, 14	83	`MC`	14
23	`STATPR`	6b, 14	86	`ME`	14
24	`X0PRT`	6a	87	`ME1`	14
25	`INITS`	3, 17	90	`PC`	14
26	`COVMAT`	10, 14	91	`AI`	14
27	`D`	4d, 14	95	`AM`	6a, 14
28	`G`	14	98	`A`	14
30	`NGCALL`	6b, 14	101	`DRADPR`	6a
31	`NITER`	14			

V symbolic subscript values, sorted numerically					
Value	*Symbol*	*Sections*	*Value*	*Symbol*	*Sections*
1	`DGNORM`	15	36	`LMAXS`	5
2	`DSTNRM`	15	37	`SCTOL`	5, 6b, 9, 15
5	`STPPAR`	15	38	`DINIT`	4b
6	`NREDUC`	15	39	`DTINIT`	4d
7	`PREDUC`	9, 15	40	`D0INIT`	4a
8	`RADIUS`	15	41	`DFAC`	4a
10	`F`	10, 15	42	`DLTFDC`	8, 9
13	`F0`	9, 15	42	`ETA0`	8, 9
17	`RELDX`	15	43	`DLTFDJ`	8, 9
31	`AFCTOL`	3, 5	44	`DELTA0`	8, 9
32	`RFCTOL`	3, 5, 9	45	`FUZZ`	16
33	`XCTOL`	3, 5	46	`RLIMIT`	16
34	`XFTOL`	5	47	`COSMIN`	16
35	`LMAX0`	7	53	`RCOND`	15

19. Fortran variations

The source code covered by this usage summary may be converted from[4] Fortran 77 to Fortran 66 by changing to a blank the ‘‘`C`’’ in column 1 of lines that come after a ‘‘`C/6`’’ line and before a line that begins ‘‘`C/7`’’ and by commenting out all lines between a line that begins ‘‘`C/7`’’ and a line that begins ‘‘`C/`’’. For example, change

4. The 1984 *Usage Summary* described conversion to Fortran 77 from Fortran 66. PORT tapes distributed since mid-1990 have been in Fortran 77 form, and the PORT source available from *netlib* has always been in Fortran 77 form.

```
C/6
C     DATA A/98/, AI/91/, AM/95/, MC/83/, ME/86/, ME1/87/, PC/90/,
C    1     PRUNIT/21/, SOLPRT/22/
C/7
      PARAMETER (A=98, AI=91, AM=95, MC=83, ME=86, ME1=87, PC=90,
     1           PRUNIT=21, SOLPRT=22)
C/
```

to

```
C/6
      DATA A/98/, AI/91/, AM/95/, MC/83/, ME/86/, ME1/87/, PC/90/,
     1     PRUNIT/21/, SOLPRT/22/
C/7
C     PARAMETER (A=98, AI=91, AM=95, MC=83, ME=86, ME1=87, PC=90,
C    1           PRUNIT=21, SOLPRT=22)
C/
```

Some modules have several sets of such lines. These changes convert some PARAMETER statements to DATA statements, turn CHARACTER variables into numeric variables, change quoted strings in DATA statements into Hollerith constants, and remove SAVE statements that may save a bit of execution time on some computers.

References

[1] J.E. Dennis, Jr., D.M. Gay, and R.E. Welsch, ‘‘An Adaptive Nonlinear Least-Squares Algorithm’’, *ACM Trans. Math. Software* **7** (1981), pp. 348–368.

[2] J.E. Dennis, Jr., D.M. Gay, and R.E. Welsch, ‘‘Algorithm 573. NL2SOL — An Adaptive Nonlinear Least-Squares Algorithm’’, *ACM Trans. Math. Software* **7** (1981), pp. 369–383.

[3] D.M. Gay, ‘‘Computing Optimal Locally Constrained Steps’’, *SIAM J. Sci. Statist. Comput.* **2** (1981), pp. 186–197.

[4] D.M. Gay, ‘‘Algorithm 611. Subroutines for Unconstrained Minimization Using a Model/Trust-Region Approach’’, *ACM Trans. Math. Software* **9** (1983), pp. 369–383.

[5] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[6] G.W. Stewart, ‘‘A Modification of Davidon’s Minimization Method to Accept Difference Approximations of Derivatives’’, *J. Assoc. Comput. Mach.* **14** (1967), pp. 72–83.