MINUIT

Function Minimization and Error Analysis

What Minuit is intended to do.

Minuit is conceived as a tool to find the minimum value of a multi-parameter function and analyze the shape of the function around the minimum. The principal application is foreseen for statistical analysis, working on chisquare or log-likelihood functions, to compute the best-fit parameter values and uncertainties, including correlations between the parameters. It is especially suited to handle difficult problems, including those which may require guidance in order to find the correct solution.

What Minuit is not intended to do

Although Minuit will of course solve easy problems faster than complicated ones, it is not intended for the repeated solution of identically parametrized problems (such as track fitting in a detector) where a specialized program will in general be much more efficient.

Submitter:
F.James, M.Roos
Language: FORTRAN 77
Language:
Library: PACKLIB
Library:

MINUIT - LONG WRITE-UP

CERN Program Library entry D506

Requests for information should be addressed to:

CERN Program Library Office
CERN-DD Division
CH-1211 Geneva 23
Switzerland
Tel. +41 22 767 4951
Fax. +41 22 767 7155
Electronic Mail address:
EARN/Bitnet: CERNLIB@CERNVM
DECnet: VXCERN::CERNLIB (node 22.190)
Internet:
CERNLIB@CERNVM.CERN.CH

Contents

Chapter 1

Minuit Basic Concepts.

1.1 The Organization of Minuit.

The Minuit package acts on a multiparameter Fortran function to which we give the generic name FCN, although the actual name may be chosen by the user. This function must be defined and supplied by the user (or by an intermediate program such as HBOOK or PAW, in case Minuit is being used under the control of such an intermediate program). The value of FCN will in general depend on one or more variable parameters whose meaning is defined by the user (or by the intermediate program), but whose trial values are determined by Minuit according to what the user requests should be done to FCN (usually minimize it).

To take a simple example, suppose the problem is to fit a polynomial through a set of data points. Then the user would write an FCN which calculates the chisquare between a polynomial and the data; the variable parameters of FCN would be the coefficients of the polynomials. Using Minuit commands, the user would request Minuit to minimize FCN with respect to the parameters, that is, find those values of the coefficients which give the lowest value of chisquare.

The user must therefore supply, in addition to the function to be analyzed, a set of commands to instruct Minuit what analysis is wanted. The commands may be given in several different forms:

- As a data file, corresponding to the traditional "data cards", for batch processing;
- Typed in at execution time at a terminal, for interactive running;
- Coded in Fortran in the calling program, which allows looping, conditional execution, and all the other possibilities of Fortran, but not interactivity, since it must be compiled before execution. This is sometimes known as running Minuit in "slave mode". HBOOK and PAW use Minuit in this way.

It is also possible to mix any of the above forms, for example starting off a fit with a standard command file, then turning it over to the interactive user for the final command steps.

1.2 Internal and External Parameters.

Each of the parameters to FCN is defined by the user as belonging to one of the following types:

Freely variable: allowed to take on any value.

Variable with limits: allowed to vary only between two limits specified by the user.

Fixed: originally defined as variable, but now taking on only the value the parameter had at the moment it was fixed, or a value later assigned by the user.

Constant: taking on only one value as specified by the user.

Undefined: never defined by user.

The user, in FCN, must of course be able to "see" all types of defined parameters, and he therefore has access to what we call the *external parameter list*, that is, the parameters as he defined them. On the other hand, the internal Minuit minimizing routines only want to "see" variable parameters without limits, and so they have access only to the *internal parameter list* which is created from the external list by the following transformation:

1. Squeeze out all parameters that are not variable.

2. Transform all variable parameters with limits, so that the transformed parameter can vary without limits. (See the next section for details concerning this transformation.) Because this transformation is non-linear, it is recommended to avoid putting limits on parameters where they are not needed.

As an example, suppose that the user has defined the following parameters:

- Parameter 1, constant.
- Parameter 3, freely variable.
- Parameter 10, variable with limits.
- Parameter 11, constant.
- Parameter 22, freely variable.
- All others undefined.

Then the internal parameter list would be as follows:

- Internal parameter 1 = external parameter 3.
- Internal parameter 2 = external parameter 10, transformed appropriately.
- Internal parameter 3 = external parameter 22.

In the above example, Minuit considers that the number of external parameters is 22 (the highest external parameter number defined), and the number of internal parameters is 3. The latter number is passed as NPAR to FCN. This is the number which determines, for example, the size of the error matrix of the parameters, since only variable parameters have errors.

An important feature of Minuit is that parameters are allowed to change types during a Minuit run. Several Minuit commands are available to make variable parameters fixed and vice-versa; to impose, change, or remove limits from variable parameters; and even to define completely new parameters at any time during a run. In addition, some Minuit routines (notably the MINOS error analysis) cause one or more variable parameters to be temporarily fixed during the calculation. Therefore, the correspondence between external and internal parameter lists is in general a dynamic one, and the value of NPAR is not necessarily constant.

1.2.1 The Transformation for Parameters with Limits.

For variable parameters with limits, Minuit uses the following transformation:

$$P_{int} = \arcsin(2\frac{P_{ext} - a}{b - a} - 1) \tag{1.1}$$

$$P_{ext} = a + \frac{b - a}{2}(\sin P_{int} + 1) \tag{1.2}$$

so that the internal value P_{int} can take on any value, while the external value P_{ext} can take on values only between the lower limit a and the upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the FCN value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if he puts limits of (0, 10^{10}) on a parameter, then the values 0.0 and 1.0 will be indistinguishable to the accuracy of most machines.

The transformation also affects the parameter error matrix, of course, so Minuit does a transformation of the error matrix (and the "parabolic" parameter errors) when there are parameter limits. Users should however realize that the transformation is only a linear approximation, and that it cannot give a meaningful result if one or more parameters is very close to a limit, where $\partial P_{ext}/\partial P_{int} \sim 0$.

For all the above reasons, it is recommended that:

- Limits on variable parameters should be used only when needed in order to prevent the parameter from taking on unphysical values.
- When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits.

Further discussion of the effects of parameter limits may be found in the last chapter.

1.3 Parameter Errors.

Minuit is usually used to find the "best" values of a set of parameters, where "best" is defined as those values which minimize a given function, FCN. The width of the function minimum, or more generally, the shape of the function in some neighbourhood of the minimum, gives information about the *uncertainty* in the best parameter values, often called by physicists the *parameter errors*. An important feature of Minuit is that it offers several tools to analyze the parameter errors.

1.3.1 FCN Normalization and the ERRor definition.

Whatever method is used to calculate the parameter errors, they will depend on the overall (multiplicative) normalization of FCN, in the sense that if the value of FCN is everywhere multiplied by a constant b, then the errors will be decreased by a factor Rb. Additive constants do not change the parameter errors, but may imply a different goodness-of-fit confidence level.

Assuming that the user knows what the normalization of his FCN means, and also that he is interested in parameter errors, the SET ERRordef command allows him to define what he means by one error, in terms of the change in FCN value which should be caused by changing one parameter by one error. If the FCN is the usual chisquare function (defined below), then ERRordef should be set to 1.0 (the default value anyway) if the user wants the usual one-standard-deviation errors. If FCN is a negative-log-likelihood function, then the one-standard-deviation value for ERRordef is 0.5. If FCN is a chisquare, but the user wants two-standard-deviation errors, then ERRordef should be = 4.0, etc.

Note that in the usual case where Minuit is being used to perform a fit to some experimental data, the parameter errors will be proportional to the uncertainty in the data, and therefore meaningful parameter errors cannot be obtained unless the measurement errors of the data are known. In the common case of a least-squares fit, FCN is usually defined as a chisquare:

$$c2(a) = \sum_{i=1}^{n} \frac{(f(x_i, a) - e_i)^2}{s_i^2} \tag{1.3}$$

where a is the vector of free parameters being fitted, and the s_i are the uncertainties in the individual measurements e_i. If these uncertainties are not known, and are simply left out of the calculation, then the fit may still have some meaning, but not the quantitative values of the resulting parameter errors. (Only the relative errors of different parameters with respect to each other may be meaningful.)

If the s_i are all overestimated by a factor b, then the resulting parameter errors from the fit will be overestimated by the same factor b.

1.3.2 The Error Matrix.

The Minuit processors MIGRAD and HESSE normally produce an error matrix. This matrix is the inverse of the matrix of second derivatives of FCN, transformed if necessary into external coordinate space[1], and multiplied by the square root of the ERRordef. Therefore, errors based on the Minuit error matrix take account of all the parameter correlations, but not the non-linearities. That is, from the error matrix alone, two-standard-deviation errors are always exactly twice as big as one-standard-deviation errors.

When the error matrix has been calculated (for example by the successful execution of a command MIGrad or HESse) then the parameter errors printed by Minuit are the square roots of the diagonal elements of this matrix. The commands SHOw COVariance and SHOw CORrelations allow the user to see the off-diagonal elements as well. The command SHOw EIGenvalues causes Minuit to calculate and print out the eigenvalues of the error matrix, which should all be positive if the matrix is positive-definite (see below on Migrad and positive-definiteness).

The effect of correlations on the individual parameter errors can be seen as follows. When parameter N is FIXed, Minuit inverts the error matrix, removes the row and column corresponding to parameter N, and re-inverts the result. The effect on the errors of the other parameters will in general be to make them smaller, since the component due to the uncertainty in parameter N has now been removed. (In the limit that a given parameter is uncorrelated with parameter N, its error will not change when parameter N is fixed.) However the procedure is not reversible, since Minuit forgets the original error matrix, so if parameter N is then RELeased, the error matrix is considered as unknown and has to be recalculated with appropriate commands.

[1] The *internal error matrix* maintained by Minuit is transformed for the user into *external coordinates*, but the numbering of rows and columns is of course still according to internal parameter numbering, since one does not want rows and columns corresponding to parameters which are not variable. The transformation therefore affects only parameters with limits; if there are no limits, internal and external error matrices are the same.

1.3.3 MINOS Errors.

The Minuit processor MINOS was probably the first, and may still be the only, generally available program to calculate parameter errors taking into account both parameter correlations and non-linearities. The MINOS error intervals are in general asymmetric, and may be expensive to calculate, especially if there are a lot of free parameters and the problem is very non-linear.

MINOS can only operate after a good minimum has already been found, and the error matrix has been calculated, so the MINOS command will normally follow a MIGRAD command. The MINOS error for a given parameter is defined as the change in the value of that parameter which causes F′ to increase by the amount UP, where F′ is the minimum of FCN with respect to all *other* free parameters, and UP is the ERRordef value specified by the user (default = 1.).

The algorithm for finding the positive and negative MINOS errors for parameter N consists of varying parameter N, each time minimizing FCN with respect to all the other NPAR-1 variable parameters, to find numerically the two values of parameter N for which the minimum of FCN takes on the values FMIN+UP, where FMIN is the minimum of FCN with respect to all NPAR parameters. In order to make the procedure as fast as possible, MINOS uses the error matrix to predict the values of all parameters at the various sub-minima which it will have to find in the course of the calculation, and in the limit that the problem is nearly linear, the predictions of MINOS will be nearly exact, requiring very few iterations. On the other hand, when the problem is very non-linear (i.e., FCN is far from a quadratic function of its parameters), that is precisely the situation when MINOS is needed in order to indicate the correct parameter errors.

1.3.4 Contour Plotting

Minuit currently offers two very different procedures for finding FCN contours. They will be identified by the corresponding command names: CONtour (which has been in Minuit for many years), and MNContour (which is new with release 89.12).

CONtour: This procedure is designed for a lineprinter or alphanumeric terminal as output device, and gives a static picture of FCN as function of the two parameters specified by the user, that is, all the other variable parameters (if any) are considered as temporarily fixed at their current values. First a range is chosen, by default two current standard deviations on either side of the current best value of each of the two parameters, and a grid size n is chosen, by default 25 by 25 positions for the full range of each parameter. Contour zero is defined as the current best function value F_{min} (presumably the minimum), and then the i^{th} contour is defined as where FCN has the value F_{min} + i^2 * UP. The procedure then simply evaluates FCN at the four corners of each of the n^2 grid positions (which makes $(n+1)^2$ evaluations) to determine whether the i^{th} contour passes through it. The method, although not very efficient or precise, is very robust, and capable of revealing unexpected multiple valleys.

MNContour: The contours calculated by MNContour are dynamic, in the sense that it represents the minimum of FCN with respect to all the other NPAR-2 parameters (if any). In statistical terms, this means that MNContour takes account of the correlations between the two parameters being plotted, and all the other variable parameters. (If this feature is not wanted, then the other parameters must be FIXed before calling MNContour.) MNContour provides the actual coordinates of the points around the contour, suitable for plotting with a graphics routine or by hand. The number of points to be calculated is chosen by the user (Default is 20 for the command-driven mode.). As a by-product, MNContour provides the MINOS errors of the two parameters in question, since these are just the extreme points of the contour (Use SHOw MINos to see them).

Chapter 2

Minuit Installation.

2.1 Minuit Releases.

Minuit has been extensively revised in 1989, but the usage is largely compatible with that of older versions which have been in use since before 1970. The *major exceptions* to this rule are:

- The traditional command-driven usage of Minuit (i.e. as invoked in one statement from the user's program) can now be accomplished by the statements:

```
EXTERNAL FCN, FUTIL
CALL MINUIT(FCN,FUTIL)
```

  where `FCN` and `FUTIL` are any allowed Fortran subroutine names, corresponding of course to the names of the user-written subroutines actually supplied, namely:

  - `FCN` is the subroutine which calculates the value of the function to be minimized or analyzed (i.e., usually the chisquare or likelihood function).
  - `FUTIL` is a Fortran function which can be called from `FCN`, and may for example be the function to which the user is fitting his data. Thus, for example, if the user wants to fit some experimental data to a Breit-Wigner shape, `FUTIL` could be a Breit-Wigner function, and `FCN` would calculate the chisquare distance between the experimental data and the function `FUTIL`. If the user does not call a function `FUTIL`, then `FUTIL` should not be declared `EXTERNAL`. **If `FUTIL` is declared `EXTERNAL`, then a subprogram `FUTIL` must be supplied in order to avoid a loader error for the unsatisfied external.**

  This new requirement to specify the names of two subroutines allows the user additional freedom, to analyze more than one different function `FCN` in one job, and/or fit to more than one different shapes `FUTIL` with the same FCN.

- The values which were previously available only directly from the COMMON blocks must now be accessed through Minuit user-callable subroutines (since the COMMON blocks have of course changed and will undoubtedly change again in the future).

- The way of indicating to Minuit that input lines (parameter definitions) are to be read from an external file, is now considerably more general and not entirely compatible with the way the old "interactive" Minuit did it.

- There are no longer any "long" and "short" versions, only a "standard" version, expected to be big enough to handle most problems, and the possibility for exceptional users to recompile their own versions for bigger problems or smaller computers.

- The format of the MINOS command is now the same as all the others.

- Several new commands and user-callable subroutines are available.

2.2 Minuit Versions.

The program is entirely in standard portable Fortran 77, and requires no external subroutines except those defined as part of the Fortran 77 standard and one logical function INTRAC[1]. The only difference between versions for different computers, apart from INTRAC, is the floating point precision (see heading below).

As with previous releases, Minuit does not use a memory manager This makes it easy to install and independent of other programs, but has the disadvantage that both the memory occupation and the maximum problem size (number of parameters) are fixed at compilation time. The old solution to this problem, which consisted of providing "long" and "short" versions, has proved to be somewhat clumsy and anyway insufficient for really exceptional users, so it is being abandoned in favour of a single "standard" version.

The currently "standard" version of Minuit will handle functions of up to 100 parameters, of which not more than 35 can be variable at one time. Because of the use of the PARAMETER statement in the Fortran source, redimensioning for larger (or smaller) versions is very easy (although it will help to have a source code manager or a good editor to propagate the modified PARAMETER statement through all the subroutines, and of course it implies recompilation). The definition of what is "standard" may well change in the light of experience, and it is likely that different installations will wish to define it differently according to their own applications. In any case, the dimensions used at compilation time are printed in the program header at execution time, and the program is of course protected against the user trying to define too many parameters. The user who finds that the version available to him is too small (or too big) must try to convince his computer manager to change the installation default or to provide an additional special version, or else he must obtain the source and recompile his own version.

2.3 Interference with Other Packages

The new Minuit has been designed to intefere as little as possible with other programs or packages which may be loaded at the same time. Thus it uses no memory manager or other external subroutines (except LOGICAL FUNCTION INTRAC), all its own subroutine names start with the letters MN (except MINUIT and the user written routines), all COMMON block names start with the characters MN7, and the user should not need to use explicitly any Minuit COMMON blocks.

In addition, more than one different functions can be minimized in the same execution module, provided the functions have different names, and provided one minimization and error analysis is completely finished before the next one begins.

2.4 New Features.

The 1988 rewrite has allowed the introduction of some new features:

- There is now the possibility of direct Fortran-callable usage, without any "input data". That is, the starting parameter values can be specified in Fortran calls, and the issuing of commands can also be performed by subroutine calls.
- The I/O has been adapted with the interactive user in mind. Input can now be given in free field format, and output can be limited to 80 columns as preferred for most terminals.
- In order to take advantage of both the Fortran-callable and interactive modes, they can also be mixed: For example, the user can set up the problem (starting parameter values) from Fortran, and then from Fortran invoke the command processor to complete the fitting and error analysis interactively.
- A new category of "SET xxx" and "SHOW xxx" commands allows the user (especially the interactive user) to interrogate and modify the Minuit environment. A (still rather primitive) HELP facility has been introduced to show the currently available possibilities.
- Minuit can be requested to read parameter definitions and/ or commands from an external file which has been prepared in advance, and external read requests can be nested.
- The user can now inform Minuit about the nature of his problem with the command: "SET STRATEGY". Strategy 1 is default; strategy 0 is intended for functions which are very expensive to evaluate; and strategy 2 is for difficult problems where additional function evaluations are needed to be sure of the accuracy of the minimum and the parameter errors.

[1] INTRAC is available from the CERN Program Library for all common computers, and in the worst case can be replaced by a LOGICAL FUNCTION returning a value of .TRUE. or .FALSE. depending on whether or not Minuit is being used interactively.

2.5 Improvements in Old Features

Most of the existing familiar Minuit routines have been rewritten and, it is hoped, improved. Known bugs and features which didn't always work, have been corrected. A good example is the part which compared user-calculated derivatives with Minuit finite differences estimates. The old version of this was so inaccurate as to be almost useless, and we believe the new version to be reliable almost down to the theoretical machine accuracy (which Minuit now knows!). Migrad has been rewritten to include a line search which improves the stability and the accuracy of the covariance matrix, and usually speeds the minimization as well, although clever users will no doubt be able to find functions which are minimized faster by the old version.

2.6 Future Improvements

A major goal of the rewriting and restructuring of Minuit has been to improve the clarity and flexibility of the program. This in turn means that future modifications will be much easier, especially the introduction of new commands, and the provision of an interface to HBOOK and PAW. The authors welcome suggestions for improvements and extensions.

2.7 Floating-point Precision

It is recommended for most applications to use 64-bit floating point precision, or the nearest equivalent on any particular machine. This means that the standard Minuit installed on Vax, IBM and Apollo will normally be the DOUBLE PRECISION version, while on CDC and Cray it will be SINGLE PRECISION.

The arguments of the user's FCN must of course correspond in type to the declarations compiled into the Minuit version being used. The same is true of course for all floating-point arguments to any Minuit routines called directly by the user in Fortran-callable mode. Furthermore, Minuit detects at execution time the precision with which it was compiled, and expects that the calculations inside FCN will be performed approximately to the same accuracy. (This accuracy is called EPSMAC and is printed in the header produced by Minuit when it begins execution.) If the user fools Minuit by using a double precision version but making internal FCN computations in single precision, Minuit will interpret roundoff noise as significant and will usually fail miserably to find a minimum. It is therefore recommended, when using double precision (REAL*8) Minuit, to make sure all FCN computations are REAL*8 by including the appropriate IMPLICIT declarations in FCN and all user subroutines called by FCN. If for some reason the computations cannot be done to a precision comparable with that expected by Minuit, the user MUST inform Minuit of this situation with the SET EPS command.

Although 64-bit precision is recommended in general, the new Minuit is so careful to use all available precision that in many cases, 32 bits will in fact be enough. It is therefore possible now to envisage in some situations (for example on microcomputers or when memory is severely limited, or if 64-bit arithmetic is very slow) the use of Minuit with 32- or 36-bit precision.

Chapter 3

How to Use Minuit

3.1 The Function FCN.

The user must always supply a Fortran subroutine which calculates the function value to be minimized or analyzed. [When Minuit is being used through an intermediate package such as HBOOK or PAW, then the FCN may be supplied by the intermediate package.] This subroutine should be of the form:

```
        SUBROUTINE FCN(NPAR,GRAD,FVAL,XVAL,IFLAG,FUTIL)
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)  !for 32-bit
machines
        DIMENSION GRAD(*),XVAL(*)
        EXTERNAL FUTIL   !(if needed and supplied by user)
C- INPUT arguments:
C-        NPAR = number of currently variable parameters.
C-        XVAL = the vector of (constant and variable) parameters.
C-        IFLAG = indicates what is to be calculated (see below).
C- OUTPUT arguments:
C-        FVAL = the function value
C-        GRAD = the (optional) vector of first derivatives.
C-
        IF (IFLAG .EQ. 1)  THEN
C           read input data,
C           calculate any necessary constants, etc.
        ENDIF
        IF (IFLAG .EQ. 2)  THEN
C
C           calculate GRAD, the first derivatives of FVAL
C           (this is optional)
        ENDIF
C             Always calculate the value of the function, FVAL,
C             which is usually a chisquare or log likelihood.
C             Optionally, calculation of FVAL may involve FTHEO = FUTIL(....)
C             It is responsibility of user to pass
C             any parameter values needed by FUTIL,
C             either through arguments, or in a COMMON block
C
         IF (IFLAG .EQ. 3)  THEN
C            will come here only after the fit is finished.
C            Perform any final calculations, output fitted data, etc.
        ENDIF
        RETURN
        END
```

The name of the subroutine may be chosen freely (in documentation we give it the generic name FCN) and must be declared EXTERNAL in the user's program which calls MINUIT (in data-driven mode) or calls Minuit subroutines (in Fortran-callable mode). The meaning of the parameters XVAL is of course defined by the user, who uses the values of those parameters to calculate his function value. The starting values must be specified

by the user (either by supplying parameter definitions from a file, or typing them at the terminal, in data-driven mode; or by calling subroutine MNPARM in Fortran-callable mode), and later values are determined by Minuit as it searches for the minimum or performs whatever analysis is requested by the user.

It is possible, by giving them different names, to analyze several different FCNs in one job. However, one analysis must be completed before the next is started. In order to avoid interference between the analyses of two different FCNs, the user should call MINUIT (in data-driven mode) or MNINIT (in Fortran-callable mode) each time a new FCN is to be studied.

3.2 Running Minuit in Data-driven Mode.

Minuit can be run in two different modes: **Data-driven mode** means that the user drives Minuit with data, either typed interactively from a terminal or from a data file in batch; and **Fortran-callable mode** means that Minuit is driven directly from Fortran subroutine calls, without data. To some extent, the two modes may also be mixed. This section describes the first mode, and is valid for both interactive and batch running. The differences between interactive and batch are described in a separate subsection below.

In *data-driven mode*, the user must supply, in addition to the Subroutine FCN, a **main program** which includes the following statements (the statements in upper case are required, those given in lower case are optional):

```
EXTERNAL FCN
external futil
call mintio(ird,iwr,isav)
CALL MINUIT(FCN,futil)
```

The name of FCN may be chosen freely, and is communicated to MINUIT as its first argument. FUTIL is the generic name of a function or subroutine which the user may optionally call from FCN, and if he does call such a routine, he must declare it external and communicate its name to MINUIT as well. If FUTIL is not used, then the second argument may be =0, and not declared EXTERNAL; if FUTIL is declared EXTERNAL, it must be supplied in the loading process.

The purpose of MINTIO is to communicate to Minuit the I/O unit numbers for input, output, and save files. If the default values of 5, 6, and 7 are acceptable, then it is not necessary to call MINTIO. It is the user's responsibility that the I/O units are properly opened for the appropriate operations.

Note: In data-driven mode, that is with CALL MINUIT, you should **not call MNINIT**, since MINUIT takes care of all initialization. To change unit numbers, call MINTIO **before calling MINUIT.**

In order that control returns to the user program after CALL MINUIT, the last command in the corresponding Data Block should be RETURN. If the last command is EXIT or STOP, then MINUIT will execute a Fortran STOP, and if the last command is END, MINUIT will read a new Data Block from the current input unit.

3.2.1 Data to drive Minuit

In data-driven mode, either interactively or in batch, Minuit reads the following data provided by the user:

- **Title** (a string of 50 characters or less) which can be chosen freely by the user, to help identify the job.
- **Parameter definitions:** for each parameter one record giving:
  1. **The parameter number**. This is the index in the array XVAL by which the user function "FCN" will access the value of the parameter.
  2. **The parameter name**. A string of ten characters to help the user in reading the Minuit output.
  3. **The starting value** of the parameter.
  4. **The starting step size**, or expected uncertainty in this parameter, if it is to be a variable parameter. Otherwise blank or zero if the value is to be constant.
  5. [Optional] The **lower bound** (limit) below which the parameter value must not vary.
  6. [Optional] The **upper bound** (limit) above which the parameter value must not vary.

  Normally the user should NOT specify limits on the parameters, that is both should be left blank. If one limit is specified, then BOTH must be specified. The properties of limits are explained elsewhere in this document.

  The format of the parameter definitions may be either fixed-field (each item in a field of width ten columns), or in free-field format. In the free-field format, items are separated by blanks or one comma, and the

parameter name must be given between single quotes. The program assumes free-field format if it finds two single quotes in the line. Parameter names will be blank-padded or truncated to be ten characters long.

- **A blank record** indicates the end of parameter definitions.
- If the user FCN reads input data from the same input stream as the Minuit data (the default stream is UNIT 5), then the FCN data should appear here.
- **Minuit commands**, which specify which actions should be performed by Minuit. Commands must not contain leading or embedded blanks, but may be truncated to three characters, and may be given in upper or lower case. Some commands have numerical arguments, and these may be given in free-field format, separated by blank(s) or one comma[1]. The list of recognized commands is given and explained below. The command HELP causes Minuit to write to the output stream a list of currently recognized commands. The command HELP SHOw lists the available SET and SHOw commands.

Any or all of the above data read by Minuit can reside on one or more different files, and Minuit can be instructed to switch to reading a different file with the SET INPUT command. Optionally, the **title** record may be preceeded by a record beginning with the characters "SET TITLE ", and the **parameter definitions** may be preceeded by a record beginning with the characters "PARAMETERS ". It is in fact recommended always to include these optional records when preparing a data file, since the file can then be read at any time (not just at the beginning of a Minuit run) and will always be interpreted correctly by Minuit.

An **example** of a typical Minuit data set:

```
SET TITLE
Fit to time distribution of K decays, Expt NA94
PARAMETERS
1 'Real(X)'  0.   .1
2 'Imag(X)'  0.   .1
5 'Delta M'  .535   .01
10 'K Short LT'  .892
11 'K Long LT'   518.3

fix 5
migrad
set print 0
minos
restore
migrad
minos
fix 5
set param  5   0.535
contour 1 2
stop
```

3.2.2 Batch and interactive running.

In its initialization phase, Minuit attempts to determine whether or not it is running interactively, by calling the logical function INTRAC, a routine in the CERN Program Library which can be provided for all commonly used computers. For our purposes, we define "running interactively" as meaning that input is coming from a terminal under the control of an intelligent being, able to make decisions based on the output he receives at the terminal. It is not always easy for INTRAC to know whether this is the case, so, depending on your operating system, Minuit can be fooled in certain cases. When this happens, the user can always override the beliefs of INTRAC with the commands SET BATch and SET INTeractive. The command SHOw INTeractive informs the user of the current mode.

According to whether or not it believes it is running interactively, Minuit behaves differently in the following ways:

- If interactive, the user is prompted before each data record is read.

[1] In older versions of Minuit, there was a special format for the MINOS command, when specifying a list of parameters; the new Minuit reads the MINOS command with the same free-field format as the other commands, so if parameter numbers are specified, they must now be separated by a blank or comma.

- If interactive, Minuit recovers from many error conditions and prompts the user to enter correct data or to specify additional required input. If the same error conditions occur in batch mode, the program either exits (if no corrective action seems possible) or ignores the incorrect data (for example, a command it cannot interpret) and continues.

- The default page size for output is a typical terminal dimension (80 by 24) if interactive, and a typical printed page size (120 by 56) if batch, but these can be overridden with the commands SET WIDth and SET LINes.

When an interactive user requests Minuit to read further input from an external file (the SET INPut command), then further input is considered to be temporarily in batch mode, until input reverts to the primary input stream.

3.3 Running Minuit in Fortran-callable mode.

The following Minuit subroutines are provided in order to allow the user to communicate with Minuit and perform all Minuit functions (define parameters, execute commands, etc.) directly from Fortran through subroutine calls. In the following list of subroutines, when a subroutine argument is defined, a right arrow ($\Rightarrow$) indicates that it is an input to the Minuit routine, and a left arrow ($\Leftarrow$) indicates an output from the Minuit routine to the user. **Note that when using the Double Precision version of Minuit (recommended for all 32-bit machines such as IBM, Vax, Apollo, etc.), all the REAL arguments given below must be declared DOUBLE PRECISION.**

- **TO INITIALIZE MINUIT:**

```
CALL MNINIT(IRD,IWR,ISAV)
```

IRD $\Rightarrow$ Unit number for input to Minuit.
IWR $\Rightarrow$ Unit number for output from Minuit.
ISAV $\Rightarrow$ Unit number for use of the SAVE command.

- **TO SPECIFY A TITLE FOR THE PROBLEM:**

```
CALL MNSETI('title')
```

title $\Rightarrow$ a string of at most 50 characters to distinguish this job or this fit.

- **TO DEFINE THE PARAMETERS**, for each parameter to be defined:

```
CHARACTER*10 CHNAM
CALL MNPARM(NUM,CHNAM,STVAL,STEP,BND1,BND2,IERFLG)
```

NUM $\Rightarrow$ parameter number as referenced by user in "FCN".
CHNAM $\Rightarrow$ name assigned by user to this parameter.
STVAL $\Rightarrow$ starting value
STEP $\Rightarrow$ starting step size or approximate parameter error.
BND1 $\Rightarrow$ lower bound (limit) on parameter value, if any (see below).
BND2 $\Rightarrow$ upper bound (limit) on parameter value, if any (see below).
IERFLG $\Leftarrow$ 0 if no error, >0 if request failed.

If BND1=BND2=0., then the parameter is considered unbounded, which is recommended unless limits are needed to make things behave well.

- **TO EXECUTE A MINUIT COMMAND:**

```
EXTERNAL FCN,FUTIL
DIMENSION ARGLIS(MAXARG)
CALL MNEXCM(FCN,'command',ARGLIS,NARG,IERFLG,FUTIL)
```

FCN $\Rightarrow$ the name of the function being analyzed,
'command' $\Rightarrow$ the Minuit command to be executed (see list below),
ARGLIS $\Rightarrow$ the list of numeric arguments to the command (if any),
NARG $\Rightarrow$ the number of arguments specified (NARG .LE. MAXARG),
IERFLG $\Leftarrow$ 0 if command was executed normally, >0 otherwise.
FUTIL $\Rightarrow$ the name of a function called by "FCN" (or =0 if not used).

Executing a command by calling MNEXCM has exactly the same effect as reading the same command in data-driven mode, except that a few commands would make no sense and are not available in Fortran-callable mode (e.g. SET INPut). The other difference is that *control always returns to the calling routine from MNEXCM*, even after commands END, EXIT, and STOP.

- **TO GET CURRENT VALUE OF A PARAMETER** for example after a fit, use the inverse of MNPARM:

```
CHARACTER*10 CHNAM
CALL MNPOUT(NUM,CHNAM,VAL,ERROR,BND1,BND2,IVARBL)
```

NUM ⇒ the number of the parameter whose value is requested,
CHNAM ⇐ the parameter name returned by Minuit,
VAL ⇐ the current parameter value (fitted value if fit has converged),
ERROR ⇐ the current estimate of parameter uncertainty (or zero if constant)
BND1 ⇐ the lower limit on parameter value, if any (otherwise zero).
BND2 ⇐ the upper limit on parameter value, if any (otherwise zero).
IVARBL ⇐ the internal parameter number if parameter is variable,
or zero if parameter is constant,
or negative if parameter is undefined.

- **TO GET CURRENT STATUS OF MINIMIZATION:**

```
CALL MNSTAT(FMIN,FEDM,ERRDEF,NPARI,NPARX,ISTAT)
```

FMIN ⇐ the best function value found so far
FEDM ⇐ the estimated vertical distance remaining to minimum
ERRDEF ⇐ the value of UP defining parameter uncertainties
NPARI ⇐ the number of currently variable parameters
NPARX ⇐ the highest (external) parameter number defined by user
ISTAT ⇐ a status integer indicating how good is the covariance matrix:
0 = not calculated at all
1 = diagonal approximation only, not accurate
2 = full matrix, but forced positive-definite
3 = full accurate covariance matrix (After MIGRAD,
this is the indication of normal convergence.)

- **TO GET CURRENT VALUE OF COVARIANCE MATRIX**

```
DIMENSION EMAT(NDIM,NDIM)
CALL MNEMAT(EMAT,NDIM)
```

EMAT ⇐ the array to be filled with the (external) covariance matrix.
NDIM ⇒ the dimensions of EMAT reserved in the user routines. NDIM should be
at least as big as the number of parameters variable at the time of the call,
otherwise the user will get only part of the full matrix.

- **TO ACCESS CURRENT PARAMETER ERRORS:**

```
CALL MNERRS(NUM,EPLUS,EMINUS,EPARAB,GLOBCC)
```

NUM ⇒ parameter number. If NUM$>$0, this is taken to be an external parameter number;
if NUM$<$0, it is the negative of an internal parameter number.
EPLUS ⇐ the positive MINOS error of parameter NUM.
EMINUS ⇐ the negative MINOS error (a negative number).
EPARAB ⇐ the "parabolic" parameter error, from the error matrix.
GLOBCC ⇐ the global correlation coefficient for parameter NUM. This is a number
between zero and one which gives the correlation between parameter
NUM and that linear combination of all other parameters which is
most strongly correlated with NUM.

Note that this call does not cause the errors to be calculated, it merely returns the current existing values. If any of the requested values has not been calculated, or has been destroyed (for example, by a redefinition of parameter values) MNERRS returns a value of zero for that argument. Thus the call to MNERRS will normally follow the execution of commands MIGRAD, HESSE, and/or MINOS.

- **TO SWITCH TO COMMAND-READING MODE** (to continue interactively for example):

```
EXTERNAL FCN, FUTIL
CALL MNINTR(FCN,FUTIL)
```

FCN $\Rightarrow$ the name of the function being analyzed
FUTIL $\Rightarrow$ the (optional) name of a function called from FCN.

The call to MNINTR will cause Minuit to read commands from the unit IRD (originally specified by the user in his call to MNINIT, IRD is usually 5 by default, which in turn is usually the terminal by default). Minuit then reads and executes commands until it encounters a command END, EXIT, RETurn, or STOP, or an end-of-file on input (or an unrecoverable error condition while reading or trying to execute a command), in which case control returns to the program which called MNINTR.

Chapter 4

Minuit Commands

In data-driven mode, Minuit accepts commands in the following format:

```
<command-string> <arg1> [arg2] etc.
```

`<command-string>` is one of the commands listed below,

`<argi>` are numerical values of *required* arguments, if any.

`[argi]` are numerical values of *optional* arguments, if any.

The arguments (if any) are separated from each other and from the command by one or more blanks or a comma. [This is now true also for the MINOS command.] Commands may be given in upper or lower case, and may be abbreviated, usually to three characters. The shortest recognized abbreviations are indicated by the capitalized part of the commands listed below. Examples of valid commands are:

```
SET INPUT  21
migrad
mig  500
SET LIMITS  14  -1.0,1.0
contours  1  2
MINOS  500  1,3,5,21,22
```

In Fortran-callable mode, all the same commands (with a few obvious exceptions as indicated) can be executed by passing the command-string and arguments to Minuit in a CALL MNEXCM statement.

The Minuit commands are:

CALl fcn <iflag>
Instructs Minuit to call subroutine "fcn" with the value of IFLAG=<iflag>. (The actual name of the subroutine called is that given by the user in his call to MINUIT or MNEXCM; the name given in this command is not used.) If <iflag> is greater than 5, Minuit assumes that a new problem is being redefined, and it forgets the previous best value of the function, covariance matrix, etc. This command can be used to instruct the user function to read new input data, recalculate constants, or otherwise modify the calculation of the function.

CLEar
Resets all parameter names and values to undefined. Must normally be followed by a PARAMETER command or equivalent, in order to define parameter values.

CONtour <par1> <par2> [devs] [ngrid]
Instructs Minuit to trace contour lines of the user function with respect to the two parameters whose external numbers are <par1> and <par2>. Other variable parameters of the function, if any, will have their values fixed at the current values during the contour tracing. The optional parameter [devs] (default value 2.) gives the number of standard deviations in each parameter which should lie entirely within the plotting area. Optional

parameter [ngrid] (default value 25 unless page size is too small) determines the resolution of the plot, i.e. the number of rows and columns of the grid at which the function will be evaluated.

END

Signals the end of a data block (i.e., the end of a fit), and implies that execution should continue, because another Data Block follows. A Data Block is a set of Minuit data consisting of (1) A Title, (2) One or more Parameter Definitions, (3) A blank line, and (4) A set of Minuit Commands. The END command is used when more than one Data Block is to be used with the same FCN function. The END command first causes Minuit to CALL FCN with IFLAG=3, in order to allow FCN to perform any calculations associated with the final fitted parameter values, unless a CALL FCN 3 command has already been executed at the current FCN value. The obsolete command **END RETurn** is the same as the **RETURN** command.

EXIT

Signals the end of execution. The EXIT command first causes Minuit to CALL FCN with IFLAG=3, in order to allow FCN to perform any calculations associated with the final fitted parameter values, unless a CALL FCN 3 command has already been executed at the current FCN value. Then it executes a Fortran STOP.

FIX <parno>

Causes parameter <parno> to be removed from the list of variable parameters, and its value will remain constant (at the current value) during subsequent minimizations, etc., until another command changes its value or its status.

HELP [SET] [SHOw]

Causes Minuit to list the available commands. The list of SET and SHOw commands must be requested separately.

HESse [maxcalls]

Instructs Minuit to calculate, by finite differences, the Hessian or error matrix. That is, it calculates the full matrix of second derivatives of the function with respect to the currently variable parameters, and inverts it, printing out the resulting error matrix. The optional argument [maxcalls] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

IMProve [maxcalls]

If a previous minimization has converged, and the current values of the parameters therefore correspond to a local minimum of the function, this command requests a search for additional distinct local minima. The optional argument [maxcalls] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

MIGrad [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, the most efficient and complete single method, recommended for general functions (see also MINImize). The minimization produces as a by-product the error matrix of the parameters, which is usually reliable unless warning messages are produced. The optional argument [maxcalls] specifies the (approximate) maximum number of function calls after which the calculation will be stopped even if it has not yet converged. The optional argument [tolerance] specifies required tolerance on the function value at the minimum. The default tolerance is 0.1, and the minimization will stop when the estimated vertical distance to the minimum (EDM) is less than 0.001*[tolerance]*UP (see SET ERR).

MINImize [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, as does the MIGrad command, but switches to the SIMplex method if Migrad fails to converge. Arguments are as for MIGrad. Note that command requires four characters to be unambiguous with MINOS.

MINOs [maxcalls] [parno] [parno] ...

Causes a Minos error analysis to be performed on the parameters whose numbers [parno] are specified. If none

are specified, Minos errors are calculated for all variable parameters. [Note that the old "special format" for the MINOS command has been abandoned.] Minos errors may be expensive to calculate, but are very reliable since they take account of non-linearities in the problem as well as parameter correlations, and are in general asymmetric. The optional argument [maxcalls] specifies the (approximate) maximum number of function calls *per parameter requested*, after which the calculation will be stopped for that parameter.

[Note: If parameters are explicitly given, only the first six have their errors calculated. If you need errors for more than six parameters, you will need to make more than one call to `MINOS`. idr 8/5/1991]

MNContour <par1> <par2> [devs] [ngrid]
This command was omitted in the original documentation. See the description of the `CONTOUR` command, and Section 1.3.4 for more details.

RELease <parno>
If <parno> is the number of a previously variable parameter which has been fixed by a command: **FIX <parno>**, then that parameter will return to variable status. Otherwise a warning message is printed and the command is ignored. Note that this command operates only on parameters which were at one time variable and have been FIXed. It cannot make constant parameters variable; that must be done by redefining the parameter with a PARAMETER command.

REStore [code]
If no [code] is specified, this command restores all previously FIXed parameters to variable status. If [code]=1, then only the last parameter FIXed is restored to variable status.

RETurn
Signals the end of a data block, and instructs MINUIT to return to the program which called it. The RETurn command first causes Minuit to CALL FCN with IFLAG=3, in order to allow FCN to perform any calculations associated with the final fitted parameter values, unless a CALL FCN 3 command has already been executed at the current FCN value. Then it executes a Fortran RETURN.

SAVe
Causes the current parameter values to be saved on a file in such a format that they can be read in again as Minuit parameter definitions. If the covariance matrix exists, it is also output in such a format. The unit number is by default 7, or that specified by the user in his call to MINTIO or MNINIT. The user is responsible for opening the file previous to issuing the SAVE command (except where this can be done interactively).

SCAn [parno] [numpts] [from] [to]
Scans the value of the user function by varying parameter number [parno], leaving all other parameters fixed at the current value. If [parno] is not specified, all variable parameters are scanned in sequence. The number of points [numpts] in the scan is 40 by default, and cannot exceed 100. The range of the scan is by default 2 standard deviations on each side of the current best value, but can be specified as from [from] to [to]. After each scan, if a new minimum is found, the best parameter values are retained as start values for future scans or minimizations. The curve resulting from each scan is plotted on the output unit in order to show the approximate behaviour of the function. This command is not intended for minimization, but is sometimes useful for debugging the user function or finding a reasonable starting point.

SEEk [maxcalls] [devs]
Causes a Monte Carlo minimization of the function, by choosing random values of the variable parameters, chosen uniformly over a hypercube centered at the current best value. The region size is by default 3 standard deviations on each side, but can be changed by specifying the value of [devs].

SET BATch
Informs Minuit that it is running in batch mode.

SET EPSmachine <accuracy>
Informs Minuit that the relative floating point arithmetic precision is <accuracy>. Minuit determines the nominal precision itself, but the SET EPS command can be used to override Minuit's own determination, when the user knows that the FCN function value is not calculated to the nominal machine accuracy. Typical values of <accuracy> are between 10^{-5} 10^{-14}.

SET ERRordef <up>
Sets the value of UP (default value= 1.), defining parameter errors. Minuit defines parameter errors as the change in parameter value required to change the function value by UP. Normally, for chisquared fits UP=1, and for negative log likelihood, UP=0.5.

SET GRAdient [force]
Informs Minuit that the user function is prepared to calculate its own first derivatives and return their values in the array GRAD when IFLAG=2 (see specification of the function "FCN"). If [force] is not specified, Minuit will calculate the FCN derivatives by finite differences at the current point and compare with the user's calculation at that point, accepting the user's values only if they agree. If [force]=1, Minuit does not do its own derivative calculation, and uses the derivatives calculated in FCN.

SET INPut [unitno] [filename]
Causes Minuit, in data-driven mode only, to read subsequent commands (or parameter definitions or title) from a different input file. If no [unitno] is specified, reading reverts to the previous input file, assuming that there was one. If [unitno] is specified, and that unit has not been opened, then Minuit attempts to open the file [filename] if a name is specified. If running in interactive mode and [filename] is not specified and [unitno] is not opened, Minuit prompts the user to enter a file name. If the word REWIND is added to the command (note: **no blanks** between 'INPUT' and 'REWIND'), the file is rewound before reading. *Note that this command is implemented in standard Fortran 77 and the results may depend on the operating system; for example, if a filename is given under VM/CMS, it must be preceeded by a slash.*

SET INTeractive
Informs Minuit that it is running interactively.

SET LIMits [parno] [lolim] [uplim]
Allows the user to change the limits on one or all parameters. If no arguments are specified, all limits are removed from all parameters. If [parno] alone is specified, limits are removed from parameter [parno]. If all arguments are specified, then parameter [parno] will be bounded between [lolim] and [uplim]. Limits can be specified in either order, Minuit will take the smaller as [lolim] and the larger as [uplim]. However, if [lolim] is equal to [uplim], an error condition results.

SET LINesperpage
Sets the number of lines that Minuit thinks will fit on one page of output. The default value is 24 for interactive mode and 56 for batch.

SET NOGradient
The inverse of SET GRAdient, instructs Minuit not to use the first derivatives calculated by the user in FCN.

SET NOWarnings
Supresses Minuit warning messages. SET WARnings is the default.

SET OUTputfile <unitno>
Instructs Minuit to write further output to unit <unitno>. (On a VAX, this is the file FORnnn.DAT. To return to terminal output use SET OUT 6 or SET OUT 5).

SET PAGethrow <integer>
Sets the carriage control character for "new page" to <integer>. Thus the value 1 produces a new page, and 0

produces a blank line, on some output devices. (see TOPofpage command)

SET PARameter <parno> <value>
Sets the value of parameter <parno> to <value>. The parameter in question may be variable, fixed, or constant, but must be defined.

SET PRIntout <level>
Sets the print level, determining how much output Minuit will produce. The allowed values and their meanings are displayed after a SHOw PRInt command, and are currently <level>=:

-1 no output except from SHOW commands
0 minimum output (no starting values or intermediate results)
1 default value, normal output
2 additional output giving intermediate results.
3 maximum output, showing progress of minimizations.

Note: See also SET WARnings command.

SET RANdomgenerator <seed>
Sets the seed of the random number generator used in SEEK. This can be any integer between 10 000 and 900 000 000, for example one which was output from a SHOw RANdom command of a previous run.

SET STRategy <level>
Sets the strategy to be used in calculating first and second derivatives and in certain minimization methods. In general, low values of <level> mean fewer function calls and high values mean more reliable minimization. Currently allowed values are 0, 1 (default), and 2.

SET TITle
Informs Minuit that the next input line is to be considered the (new) title for this task or sub-task. This is for the convenience of the user in reading his output. This command is available only in data-driven mode; in Fortran-callable mode use CALL MNSETI.

SET WARnings
Instructs Minuit to output warning messages when suspicious conditions arise which may indicate unreliable results. This is the default.

SET WIDthpage
Informs Minuit of the output page width. Default values are 80 for interactive jobs and 120 for batch.

SHOw XXXX
All SET XXXX commands have a corresponding SHOw XXXX command. In addition, the SHOw commands listed starting here have no corresponding SET command for obvious reasons. The full list of SHOw commands is printed in response to the command HELP SHOw.

SHOw CORrelations
Calculates and prints the parameter correlations from the error matrix.

SHOw COVariance
Prints the (external) covariance (error) matrix.

SHOw EIGenvalues
Calculates and prints the eigenvalues of the covariance matrix.

SHOw FCNvalue
Prints the current value of FCN.

SHOw MINoserrors
Prints the current minos errors.

SIMplex [maxcalls] [tolerance]
Performs a function minimization using the simplex method of Nelder and Mead. Minimization terminates either when the function has been called (approximately) [maxcalls] times, or when the estimated vertical distance to minimum (EDM) is less than [tolerance]. The default value of [tolerance] is 0.1*UP (see SET ERR).

STAndard
Causes Minuit to execute the Fortran instruction **CALL STAND** where STAND is a subroutine supplied by the user.

STOP
Same as EXIT.

TOPofpage
Causes Minuit to write the character specified in a SET PAGethrow command (default = “1”) to column 1 of the output file, which may or may not position your output medium to the top of a page depending on the device and system.

Chapter 5

How to get the right answer from Minuit.

The goal of Minuit - to be able to minimize and analyze parameter errors for all possible user functions with any number of variable parameters - is of course impossible to realise, even in principle, in a finite amount of time. In practice, some assumptions must be made about the behaviour of the function in order to avoid evaluating it at all possible points. In this chapter we give some hints on how the user can help Minuit to make the right assumptions.

5.1 Which Minimizer to Use.

One of the historically interesting advantages of Minuit is that it was probably the first minimization program to offer the user a choice of several minimization algorithms. This could be taken as a reflection of the fact that none of the algorithms known at that time were good enough to be universal, so users were encouraged to find the one that worked best for them. Since then, algorithms have improved considerably, but Minuit still offers several, mostly so that old users will not feel cheated, but also to help the occasional user who does manage to defeat the best algorithms. Minuit currently offers five commands which can be used to find a smaller function value, in addition to a few others, like MINOS and IMPROVE, which will retain a smaller function value if they stumble on one unexpectedly (or, in the case of IMPROVE, hopefully). The commands which can be used to minimize are:

- **MIGRAD**. This is the best minimizer for nearly all functions. It is a variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness. It will run faster if you SET STRategy 0 and will be more reliable if you SET STRategy 2 (although the latter option may not help much). Its main weakness is that it depends heavily on knowledge of the first derivatives, and fails miserably if they are very inaccurate. If first derivatives are a problem, they can be calculated analytically inside FCN (see elsewhere in this writeup) or if this is not feasible, the user can try to improve the accuracy of Minuit's numerical approximation by adjusting values using the SET EPS and/or SET STRategy commands (see Floating Point Precision and SET STRategy).

- **MINIMIZE**. This is equivalent to MIGRAD, except that if MIGRAD fails, it reverts to SIMPLEX and then calls MIGRAD again. This is what the old MIGRAD command used to do, but it was removed from the MIGRAD command so that users would have a choice, and because it is seldom of any use to call SIMPLEX when MIGRAD has failed (there are of course exceptions).

- **SCAN**. This is not intended to minimize, and just scans the function, one parameter at a time. It does however retain the best value after each scan, so it does some sort of highly primitive minimization.

- **SEEK**. We have retained this Monte Carlo search mainly for sentimental reasons, even though the limited experience with it is less than spectacular. The method now incorporates a Metropolis algorithm which always moves the search region to be centred at a new minimum, and has probability $e^{(-F/F_{min})}$ of moving the search region to a higher point with function value F. This gives it the theoretical ability to jump through function barriers like a multidimensional quantum mechanical tunneler in search of isolated minima, but it is widely believed by at least half of the authors of Minuit that this is unlikely to work in practice (counterexamples are welcome) since it seems to depend critically on choosing the right average step size for the random jumps, and if you knew that, you wouldn't need Minuit.

- **SIMPLEX**. This genuine multidimensional minimization routine is usually much slower than MIGRAD, but it does not use first derivatives, so it should not be so sensitive to the precision of the FCN calculations, and is even rather robust with respect to gross fluctuations in the function value. However, it gives no

reliable information about parameter errors, no information whatsoever about parameter correlations, and worst of all cannot be expected to converge accurately to the minimum in a finite time. Its estimate of EDM is largely fantasy, so it would not even know if it did converge.

5.2 Floating point Precision

Minuit figures out at execution time the precision with which it was compiled, and assumes that FCN provides about the same precision. That means not just the length of the numbers used and returned by FCN, but the actual mathematical accuracy of the calculations. The section on Floating point Precision in Chapter One describes what to do if this is not the case.

5.3 Parameter Limits

Putting limits (absolute bounds) on the allowed values for a given parameter, causes Minuit to make a non-linear transformation of its own internal parameter values to obtain the (external) parameter values passed to FCN. To understand the adverse effects of limits, see "The Transformation for Parameters with Limits" in Chapter 1. Basically, the use of limits should be avoided unless needed to keep the parameter inside a desired range.

If parameter limits are needed, in spite of the effects described in Chapter One, then the user should be aware of the following techniques to alleviate problems caused by limits:

5.3.1 Getting the Right Minimum with Limits.

If MIGRAD converges normally to a point where no parameter is near one of its limits, then the existence of limits has probably not prevented Minuit from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimizer has become "blocked" at a limit. This may normally happen only if the parameter is so close to a limit (internal value at an odd multiple of $\pm\pi/2$) that Minuit prints a warning to this effect when it prints the parameter values.

The minimizer can become blocked at a limit, because at a limit the the derivative seen by the minimizer:

$$\frac{\partial F}{\partial P_{int}} = \frac{\partial F}{\partial P_{ext}} \frac{\partial P_{ext}}{\partial P_{int}} = \frac{\partial F}{\partial P_{ext}} 0 = 0 \qquad (5.1)$$

is zero no matter what the real derivative $\partial F/\partial P_{ext}$ is. For a stepping method (like SIMPLEX) this seldom poses any problem, but a method based on derivatives (MIGRAD) may become blocked at such a value. If this happens, it may be necessary to move the value of the parameter in question a significant distance from the limit (with SET PARam) and restart the minimization, perhaps with that parameter fixed temporarily. We are investigating ways to induce Minuit to extricate itself from such situations automatically, but it is not so obvious as it seems, and for the moment must sometimes be done by hand.

5.3.2 Getting the right parameter errors with limits.

In the best case, where the minimum is far from any limits, Minuit will correctly transform the error matrix, and the parameter errors it reports should be accurate and very close to those you would have got without limits. In other cases (which should be more common, since otherwise you wouldn't need limits), the very meaning of parameter errors becomes problematic. Mathematically, since the limit is an absolute constraint on the parameter, a parameter at its limit has no error, at least in one direction. The error matrix, which can assign only symmetric errors, then becomes essentially meaningless. On the other hand, the MINOS analysis is still meaningful, at least in principle, as long as MIGRAD (which is called internally by MINOS) does not get blocked at a limit. Unfortunately, the user has no control over this aspect of the MINOS calculation, although it is possible to get enough printout from the MINOS command to be able to determine whether the results are reliable or not.

5.4 Fixing and Releasing Parameters

When Minuit needs to be guided to the "right" minimum, often the best way to do this is with the FIX and RELEASE commands. That is, suppose you have a problem with ten free parameters, and when you minimize with respect to all at once, Minuit goes to an unphysical solution characterized by an unphysical or unwanted value of parameter number four. One way to avoid this is to FIX parameter four at a "good" value (not

necessarily the best, since you presumably don't know that yet), and minimize with respect to the others. Then RELease 4 and minimize again. If the problem admits a "good" physical solution, you will normally find it this way. If it doesn't work, you may see what is wrong by the following sequence (where 'xxx' is the expected physical value for parameter four):

```
SET PARAM 4 xxx
FIX 4
MIGRAD
RELEASE 4
SCAN 4
```

where the SCAN command gives you a picture of FCN as a function of parameter four alone, the others being fixed at their current best values. If you suspect the difficulty is due to parameter five, then add the command

```
CONTOUR  4  5
```

to see a two-dimensional picture.

5.5 Interpretation of Parameter Errors

There are two kinds of problems that can arise: The **reliability** of Minuit's error estimates, and their **statistical interpretation**, assuming they are accurate.

5.5.1 Statistical Interpretation.

For discussuion of basic concepts, such as the meaning of the elements of the error matrix, parabolic versus MINOS errors, the appropriate value for UP (see SET ERRdef), and setting of exact confidence levels, see (in order of increasing complexity and completeness):

- *Interpretation of the Errors on Parameters*, available from the CERN Program Library as a supplement to this write-up.
- *Determining the Statistical Significance of Experimental Results*, by F. James. CERN DD Report DD/81/02, also printed in the proceedings of the 1980 CERN School of Computing (CERN Yellow Report 81-03).
- *Statistical Methods in Experimental Physics*, by Eadie et al., North-Holland 1971.

5.5.2 The Reliability of Minuit Error Estimates.

Minuit always carries around its own current estimates of the parameter errors, which it will print out on request, no matter how accurate they are at any given point in the execution. For example, at initialization, these estimates are just the starting step sizes as specified by the user. After a MIGRAD or HESSE step, the errors are usually quite accurate, unless there has been a problem. Minuit, when it prints out error values, also gives some indication of how reliable it thinks they are. For example, those marked 'CURRENT GUESS ERROR' are only working values not to be believed, and 'APPROXIMATE ERROR' means that they have been calculated but there is reason to believe that they may not be accurate. If no mitigating adjective is given, then at least Minuit believes the errors are accurate, although there is always a small chance that Minuit has been fooled. Some visible signs that Minuit may have been fooled are:

- Warning messages produced during the minimization or error analysis.
- Failure to find new minimum.
- Value of EDM too big. For a "normal" minimization, after MIGRAD, the value of EDM is usually more than three orders of magnitude smaller than UP (the ERRor def), unless a looser tolerance has been specified.
- Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others.
- Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parametrized so that individual errors are not very meaningful because they are so highly correlated.

- Parameter at limit. This condition, signalled by a Minuit warning message, may make both the function minimum and parameter errors unreliable. See the discussion above *Getting the right parameter errors with limits*.

The best way to be absolutely sure of the errors, is to use "independent" calculations and compare them, or compare the calculated errors with a picture of the function if possible. For example, if there is only one free parameter, the command SCAN allows the user to verify approximately the function curvature. Similarly, if there are only two free parameters, use CONTOUR. To verify a full error matrix, compare the results of MIGRAD with those (calculated afterward) by HESSE, which uses a different method. And of course the most reliable and most expensive technique, which must be used if asymmetric errors are required, is MINOS.

5.6 Convergence in MIGRAD, and Positive-definiteness.

MIGRAD uses its current estimate of the covariance matrix of the function to determine the current search direction, since this is the optimal strategy for quadratic functions and "physical" functions should be quadratic in the neighbourhood of the minimum at least. The search directions determined by MIGRAD are guaranteed to be downhill only if the covariance matrix is positive-definite, so in case this is not true, it makes a positive-definite approximation by adding an appropriate constant along the diagonal as determined by the eigenvalues of the matrix. Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum, although it may not be so for all points far away from the minimum, even for a well-determined physical problem. Therefore, if MIGRAD reports that it has found a non-positive-definite covariance matrix, this may be a sign of one or more of the following:

- **A non-physical region**. On its way to the minimum, MIGRAD may have traversed a region which has unphysical behaviour, which is of course not a serious problem as long as it recovers and leaves such a region.
- **An underdetermined problem**. If the matrix is not positive-definite even at the minimum, this may mean that the solution is not well-defined, for example that there are more unknowns than there are data points, or that the parametrization of the fit contains a linear dependence. If this is the case, then Minuit (or any other program) cannot solve your problem uniquely, and the error matrix will necessarily be largely meaningless, so the user must remove the underdeterminedness by reformulating the parametrization. Minuit cannot do this itself, but it can provide some hints (contours, global correlation coefficients, eigenvalues) which can help the clever user to find out what is wrong.
- **Numerical inaccuracies**. It is possible that the apparent lack of positive-definiteness is in fact only due to excessive roundoff errors in numerical calculations, either in FCN or in Minuit. This is unlikely in general, but becomes more likely if the number of free parameters is very large, or if the parameters are badly scaled (not all of the same order of magnitude), and correlations are also large. In any case, whether the non-positive-definiteness is real or only numerical is largely irrelevant, since in both cases the error matrix will be unreliable and the minimum suspicious.

5.7 Additional Trouble-shooting

When Minuit just doesn't work, some of the more common causes are:

- **Precision mismatch**. Make sure your FCN has been compiled with the same precision as the version of Minuit you are using. When using DOUBLE PRECISION, it is safest to use the IMPLICIT declaration to make sure that everything is DOUBLE PRECISION, not just the arguments of FCN but also the internal variables. Note that depending on the computer system used, floating-point constants may be passed as single precision in subroutine arguments, even if there is an IMPLICIT DOUBLE PRECISION statement (which is strictly speaking correct since the IMPLICIT statement refers only to variables, not constants). Therefore, if constants are used as arguments in subroutine calls, they must be explicitly of the right precision (for example, on Apollo, even 0. is not equal to 0.D0).

  If the problem is only one of precision, and not of word length mismatch, an appropriate SET EPS command may fix it.
- **Trivial bugs in FCN**. The possibilities for Fortran bugs are numerous. Probably the most common among physicists inexperienced in Fortran is the confusion between REAL and INTEGER types, which you can sometimes get away with, but not always. [For example, if A and B are REAL variables, the Fortran statement A = 2*B is not good programming, but happens to do what the user probably intended, whereas

the statement A = B + 2/3 almost certainly will not do what the user intended.] Minuit can spot some trivial bugs itself, and issues a warning when it detects an unusual FCN behaviour. Such a warning should be taken seriously.

Minuit also offers some tools (especially SCAN) which can help the user to find trivial bugs.

- **Overwriting in a user routine**. Overwriting most often occurs when setting the values of a local array or an array in COMMON, and elements outside the dimensions of the array are addressed. Most computer systems do not detect this error unless you attempt to write into a protected area of memory, and of course Minuit is also helpless, especially if Minuit itself is being overwritten. The symptoms of user overwriting may be almost anything, including unusual behaviour of Minuit itself. The effects depend critically on where instructions and data are loaded in memory, so they may change completely if the same program is recompiled with different compiler options or reloaded in a different sequence, even though the compiler and loader are not at fault.

- **Changing the values of input arguments**. In subroutine FCN, for example, the arguments NPAR and IFLAG, as well as the values of the parameters themselves, are only input to FCN and their values should not be changed inside FCN. Minuit is now protected against this in principle, since the user only gets a copy of the value, not the actual address of the internal Minuit variable, but still this is a symptom of misunderstanding by the user.

  If you really want to change the number of variable parameters, this must be done with commands like FIX and RELEASE, or by redefining parameters using command PARAMETER or CLEAR.

  Similarly, if a parameter takes on an unwanted value, it will do no good to change its value inside FCN: In the best case, Minuit won't see your improved value, and in the worst case, it will produce unpredictable results. To set a parameter to a certain value, use the command SET PARam, and to keep it within certain bounds, use the command SET LIMits. If the parameter must obey more complicated constraints, you must find a trick such as adding a penalty value to FCN outside of the physical region, to force it back to where you want it.

- **An ill-posed problem**. For questions of parameter dependence, see the discussion above on positive-definiteness. Other mathematical problems which can arise are: **excessive numerical roundoff** – be espcially careful of exponential and factorial functions which get big very quickly and lose accuracy; **starting too far from the solution** – the function may have unphysical local minima, especially at infinity in some variables; **incorrect normalization** – in likelihood functions, the probability distributions must be normalized or at least have an integral which is independent of the values of the variable parameters.

- **A bug in Minuit**. This is extremely unlikely, but it did happen once. If a bug is suspected, and all other possible causes can be eliminated, please try to save a copy of the input and output files, listing of FCN, and other information that may be relevant, and send them to JAMES at CERNVM or VXCERN::JAMES or JAMES on the Cern Apollo Domain.

Chapter 6

EXAMPLES

We give here one full example of a real fit, performed first in batch data-driven mode, then the same fit performed by Fortran calls.

6.1 A data-driven fit

The example job given here is set up for batch processing. The OPEN statements assign the input and output files, and are somewhat computer-dependent (those given here are for a Vax). On many systems, it may be more convenient (or necessary) to perform the file assignments in JCL rather than from the Fortran, but whatever the user decides, the files must be opened and the unit numbers communicated to Minuit before the call to MINUIT.

The same job could be run interactively, in which case the input and output files would be assigned to the terminal, and the "user's data" listed below, instead of coming from a file, would be typed in directly to the terminal.

6.1.1 The User's main program

```
      PROGRAM DSDQ
      EXTERNAL FCNKO
      OPEN (UNIT=5,FILE='DSDQ.DAT',STATUS='OLD')
      OPEN
(UNIT=6,FILE='DSDQ.OUT',STATUS='NEW',FORM='FORMATTED')
CC      CALL MINTIO(5,6,7)   !not needed, default values
      CALL MINUIT(FCNKO,0)
      STOP
      END
```

6.1.2 The User's FCN

```
      SUBROUTINE FCNKO(NPAR,GIN,F,X,IFLAG)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      REAL THPLUI, THMINI
      DIMENSION X(*),GIN(*)
      PARAMETER (MXBIN=50)
      DIMENSION THPLU(MXBIN),THMIN(MXBIN),T(MXBIN),
     +    EVTP(MXBIN),EVTM(MXBIN)
      DATA  NBINS,NEVTOT/ 30,250/
      DATA (EVTP(IGOD),IGOD=1,30)
     +         /11.,  9., 13., 13., 17.,  9.,  1.,  7.,  8., 9.,
     +           6.,  4.,  6.,  3.,  7.,  4.,  7.,  3.,  8., 4.,
     +           6.,  5.,  7.,  2.,  7.,  1.,  4.,  1.,  4., 5./
      DATA (EVTM(IGOD),IGOD=1,30)
     +         / 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., 1.,
     +           0.,  2.,  1.,  4.,  4.,  2.,  4.,  2.,  2., 0.,
     +           2.,  3.,  7.,  2.,  3.,  6.,  2.,  4.,  1., 5./
C
      XRE = X(1)
```

```
      XIM = X(2)
      DM = X(5)
      GAMS = 1.0/X(10)
      GAML = 1.0/X(11)
      GAMLS = 0.5*(GAML+GAMS)
      IF (IFLAG .NE. 1)  GO TO 300
C                        generate random data
      STHPLU = 0.
      STHMIN = 0.
      DO 200 I= 1, NBINS
      T(I) = 0.1*REAL(I)
      TI = T(I)
      EHALF = EXP(-TI*GAMLS)
      TH =      ((1.0-XRE)**2 + XIM**2) * EXP(-TI*GAML)
      TH = TH + ((1.0+XRE)**2 + XIM**2) * EXP(-TI*GAMS)
      TH = TH -               4.0*XIM*SIN(DM*TI) * EHALF
      STERM = 2.0*(1.0-XRE**2-XIM**2)*COS(DM*TI) * EHALF
      THPLU(I) = TH + STERM
      THMIN(I) = TH - STERM
      STHPLU = STHPLU + THPLU(I)
      STHMIN = STHMIN + THMIN(I)
  200 CONTINUE
      NEVPLU = REAL(NEVTOT)*(STHPLU/(STHPLU+STHMIN))
      NEVMIN = REAL(NEVTOT)*(STHMIN/(STHPLU+STHMIN))
      WRITE (6,'(A)') '  LEPTONIC K ZERO DECAYS'
      WRITE (6,'(A,3I10)') ' PLUS, MINUS, TOTAL=',NEVPLU,NEVMIN,NEVTOT
      WRITE (6,'(A)')
     +  '0    TIME        THEOR+      EXPTL+     THEOR-EXPTL-'
      SEVTP = 0.
      SEVTM = 0.
      DO 250 I= 1, NBINS
      THPLU(I) = THPLU(I)*REAL(NEVPLU) / STHPLU
      THMIN(I) = THMIN(I)*REAL(NEVMIN) / STHMIN
      THPLUI = THPLU(I)
CCCCC       remove the CCC to generate random data
CCC      CALL POISSN(THPLUI,NP,IERROR)
CCC      EVTP(I) = NP
      SEVTP = SEVTP + EVTP(I)
      THMINI = THMIN(I)
CCC      CALL POISSN(THMINI,NM,IERROR)
CCC      EVTM(I) = NM
      SEVTM = SEVTM + EVTM(I)
      IF (IFLAG .NE. 4)
     + WRITE (6,'(1X,5G12.4)') T(I),THPLU(I),EVTP(I),THMIN(I),EVTM(I)
  250 CONTINUE
      WRITE (6, '(A,2F10.2)') ' DATA EVTS PLUS, MINUS=', SEVTP,SEVTM
C                       calculate chisquare
  300 CONTINUE
      CHISQ = 0.
      STHPLU = 0.
      STHMIN = 0.
      DO 400 I= 1, NBINS
      TI = T(I)
      EHALF = EXP(-TI*GAMLS)
      TH =      ((1.0-XRE)**2 + XIM**2) * EXP(-TI*GAML)
      TH = TH + ((1.0+XRE)**2 + XIM**2) * EXP(-TI*GAMS)
      TH = TH -               4.0*XIM*SIN(DM*TI) * EHALF
      STERM = 2.0*(1.0-XRE**2-XIM**2)*COS(DM*TI) * EHALF
      THPLU(I) = TH + STERM
      THMIN(I) = TH - STERM
```

```
      STHPLU = STHPLU + THPLU(I)
      STHMIN = STHMIN + THMIN(I)
  400 CONTINUE
      THP = 0.
      THM = 0.
      EVP = 0.
      EVM = 0.
      IF (IFLAG .NE. 4) WRITE (6,'(1H0,10X,A,20X,A)')
     +  'POSITIVE LEPTONS','NEGATIVE LEPTONS'
      IF (IFLAG .NE. 4) WRITE (6,'(A,3X,A)')
     +    '      TIME    THEOR    EXPTL    CHISQ',
     +    '      TIME    THEOR    EXPTL    CHISQ'
C
      DO 450 I= 1, NBINS
      THPLU(I) = THPLU(I)*SEVTP / STHPLU
      THMIN(I) = THMIN(I)*SEVTM / STHMIN
      THP = THP + THPLU(I)
      THM = THM + THMIN(I)
      EVP = EVP + EVTP(I)
      EVM = EVM + EVTM(I)
C  Sum over bins until at least four events found
      IF (EVP .GT. 3.)  THEN
         CHI1 = (EVP-THP)**2/EVP
         CHISQ = CHISQ + CHI1
         IF (IFLAG .NE. 4)
     +      WRITE (6,'(1X,4F9.3)') T(I),THP,EVP,CHI1
         THP = 0.
         EVP = 0.
      ENDIF
      IF (EVM .GT. 3)  THEN
         CHI2 = (EVM-THM)**2/EVM
         CHISQ = CHISQ + CHI2
         IF (IFLAG .NE. 4)
     +      WRITE (6,'(42X,4F9.3)') T(I),THM,EVM,CHI2
         THM = 0.
         EVM = 0.
      ENDIF
  450 CONTINUE
      F = CHISQ
      RETURN
      END
```

6.1.3 The user's data to drive Minuit.

```
set title
FIT DELTA S/ DELTA Q RULE TO LEPTONIC K ZERO DECAYS
parameters
1 'Real(X)' 0. .1
2 'Imag(X)' 0. .1
5 'Delta M'  .535 .01
10 'K Short LT' .892
11 'K Long LT'   518.3

fix 5
migr
print 0
set print 0
minos
restore
migrad
```

```
minos
set param 5 0.535
fix 5
contour 1 2
stop
```

6.2 The same example in Fortran-callable mode.

6.2.1 The User's main program and subroutine

(this takes the place of the data in the above example).

```
      PROGRAM DSDQ
C             Minuit test case.  Fortran-callable.
C             Fit randomly-generated leptonic K0 decays to the
C       time distribution expected for interfering K1 and K2,
C       with free parameters Re(X), Im(X), DeltaM, and GammaS.
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      EXTERNAL FCNK0
CC    OPEN (UNIT=6,FILE='DSDQ.OUT',STATUS='NEW',FORM='FORMATTED')
      DIMENSION NPRM(5),VSTRT(5),STP(5),ARGLIS(10)
      CHARACTER*10 PNAM(5)
      DATA NPRM /   1   ,    2   ,     5    ,   10     ,  11/
      DATA PNAM /'Re(X)', 'Im(X)', 'Delta M','T Kshort','T Klong'/
      DATA VSTRT/   0.  ,    0.  ,    .535  ,   .892   ,  518.3/
      DATA STP  /   0.1 ,    0.1 ,     0.1  ,     0.   ,   0./
C        Initialize Minuit, define I/O unit numbers
      CALL MNINIT(5,6,7)
C        Define parameters, set initial values
      ZERO = 0.
      DO 11  I= 1, 5
       CALL MNPARM(NPRM(I),PNAM(I),VSTRT(I),STP(I),ZERO,ZERO,IERFLG)
       IF (IERFLG .NE. 0)  THEN
          WRITE (6,'(A,I)')  ' UNABLE TO DEFINE PARAMETER NO.',I
          STOP
       ENDIF
   11 CONTINUE
C
      CALL MNSETI('Time Distribution of Leptonic K0 Decays')
C       Request FCN to read in (or generate random) data (IFLAG=1)
          ARGLIS(1) = 1.
      CALL MNEXCM(FCNK0, 'CALL FCN', ARGLIS ,1,IERFLG)
C
         ARGLIS(1) = 5.
      CALL MNEXCM(FCNK0,'FIX', ARGLIS ,1,IERFLG)
         ARGLIS(1) = 0.
      CALL MNEXCM(FCNK0,'SET PRINT', ARGLIS ,1,IERFLG)
      CALL MNEXCM(FCNK0,'MIGRAD', ARGLIS ,0,IERFLG)
      CALL MNEXCM(FCNK0,'MINOS', ARGLIS ,0,IERFLG)
         CALL PRTERR
         ARGLIS(1) = 5.
      CALL MNEXCM(FCNK0,'RELEASE', ARGLIS ,1,IERFLG)
      CALL MNEXCM(FCNK0,'MIGRAD', ARGLIS ,0,IERFLG)
      CALL MNEXCM(FCNK0,'MINOS', ARGLIS ,0,IERFLG)
         ARGLIS(1) = 3.
      CALL MNEXCM(FCNK0,'CALL FCN', ARGLIS , 1,IERFLG)
         CALL PRTERR
      CALL MNEXCM(FCNK0,'STOP ', 0,0,IERFLG)
      STOP
      END
```

```
      SUBROUTINE PRTERR
C   a little hand-made routine to print out parameter errors
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C  find out how many variable parameters there are
      CALL MNSTAT(FMIN,FEDM,ERRDEF,NPARI,NPARX,ISTAT)
C   and their errors
      DO 50 I= 1, NPARI
      CALL MNERRS(-I,EPLUS,EMINUS,EPARAB,GLOBCC)
      WRITE (6,45) I,EPLUS,EMINUS,EPARAB,GLOBCC
   45 FORMAT (5X,I5,4F12.6)
   50 CONTINUE
      RETURN
      END
```

6.2.2 The User's FCN

The FCN is exactly the same in Fortran-callable mode as in data-driven mode.