MERLIN–3.0
User Manual

D.G. Papageorgiou, I.N. Demetropoulos
Department of Chemistry

I.E. Lagaris
Department of Computer Science

http://nrt.cs.uoi.gr/merlin

UNIVERSITY OF IOANNINA
GREECE

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What kind of problems Merlin handles

Multidimensional minimization is a common procedure needed in many fields. A variety of problems in engineering, physics, chemistry, etc., are frequently reduced to ones of minimizing a function of many variables. For instance we refer to systems of non–linear equations, to variational methods, to curve fitting and to the training of neural networks. Minimizing a multidimensional function faces a lot of difficulties. There is no single method that can tackle all problems in a satisfactory way. It has been realized that one needs a strategy, combining different methods, to efficiently handle a wide spectrum of problems. The presence of constraints, even of simple ones, enhances the difficulty. Many algorithms require evaluation of the gradient. This imposes additional problems since it is not always straightforward to code it. Hence one resorts to approximating the derivatives using differencing, that costs in computing time as well as in accuracy.

Merlin is an integrated environment designed to solve optimization problems. It is devised to be easy–to–use, and implemented so as to be portable among different platforms. Another feature is that Merlin is *open*, i.e. a *plug-in* mechanism is provided so that others can easily embed their own code modules. Merlin handles the following category of problems:
Find a local minimum of the function:

$$f(\boldsymbol{x}), \quad \boldsymbol{x} \in R^N \quad \boldsymbol{x} = [x_1, x_2, \ldots, x_N]^T$$

under the conditions:

$$x_i \in [l_i, u_i] \quad \text{for} \quad i = 1, 2, \ldots, N$$

Special merit has been taken for problems where the objective function can be written as sum of squares i.e.:

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} f_i^2(\boldsymbol{x})$$

This form is particularly suited when one needs to fit data points using a model function. One then minimizes the chisquare which is of the above form. In this case MERLIN can calculate parabolic estimates of the confidence intervals for the model parameters as well as partial covariance matrices.

MERLIN can be used both interactively and in batch. Interactively the user drives MERLIN by entering commands through the keyboard. In batch MERLIN reads commands from an input command file. Interactively MERLIN is tolerant to errors in input and issues appropriate warning messages, while in batch aborts. There are various commands at the user's disposal that either invoke minimization algorithms or perform other auxiliary operations.

MERLIN is programmable. Its programming language MCL (Merlin Control Language), is a high level, easy to learn language. The MCL compiler takes as input a strategy (coded in MCL) and produces as output a file that contains commands that can steer MERLIN appropriately. MERLIN and MCL are both written in ANSI Fortran–77 to guarantee portability.

1.2 Conventions

1.2.1 Typing

- Lower case boldface letters ($\boldsymbol{x}$, $\boldsymbol{g}$, etc.) stand for vectors in the N–dimensional space.
- Upper case boldface letters ($\boldsymbol{G}$, $\boldsymbol{H}$, etc.) stand for $N \times N$ matrices.
- MERLIN commands, Fortran code and I/O data are printed using a monospaced font.
- $\langle$cr$\rangle$ denotes keying the "carriage return".
- ... implies that preceding symbols may be repeated.
- Optional items are enclosed in square brackets [...]

1.2.2 Symbols

- N, n The number of parameters (dimensionality of the problem).
- $f(\boldsymbol{x})$ The objective function.
- x_i The i^{th} component of $\boldsymbol{x}$.
- $f_k(\boldsymbol{x})$ The k^{th} term entering in the calculation of the objective function when it has the form: $\sum_{k=1}^{M} f_k^2(\boldsymbol{x})$
- M The number of the squared terms involved in the above sum.

- $\boldsymbol{g}$ The gradient vector of the objective function $\nabla f(\boldsymbol{x})$.
- $\boldsymbol{J}$ The Jacobian matrix with $J_{ij} = \frac{\partial f_i}{\partial x_j}$.
- $\boldsymbol{G}$ The Hessian matrix with $G_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$.
- $\boldsymbol{B}$ An approximation to $\boldsymbol{G}$.
- $\boldsymbol{H}$ An approximation to $\boldsymbol{G}^{-1}$.
- l_i, u_i The lower and upper bounds for x_i, referred to as the left and right margins.
- η The machine accuracy, i.e. the approximately largest constant such that in machine precision $1 + \eta$ is equal to 1.

Chapter 2

Installation and configuration

2.1 The pieces

MERLIN is under continuous development. The authors make several minor revisions from time to time that enhance its performance and functionality. However these revisions alone do not justify a new publication. For this reason the latest revision of the MERLIN–MCL package is available from: `http://nrt.cs.uoi.gr/merlin`.

The complete distribution contains the following files:

- `install.f` The MERLIN–MCL installation program.
- `merlin.d` Source code for the MERLIN package. Does not contain any of the user written routines. Empty plug–in modules for extending MERLIN's functionality are included here.
- `sample.d` Sample user written routines: A sample main program, and the source code for the Rosenbrock function in both forms, general and sum of squares. It also contains the source for the gradient, Hessian and Jacobian.
- `HELP` The MERLIN help file.
- `PDESC` The panel description file. It used by both MERLIN and MCL, in order to correctly recognize panel commands.
- `mcl.d` Source code for the MCL compiler.
- `manual.ps` The present manual in PostScript form.

Note that the files `merlin.d`, `sample.d` and `mcl.d` must be processed by the installer program before they can be actually compiled. The procedure is explained in the following sections.

2.2 Installation

2.2.1 Why an installation procedure

The MERLIN–MCL source code contains a number of parameters and options that must be customized in order for the programs to operate correctly. Some of them can be set at run–time while others must be hard–coded in the Fortran source. An example of the later is the maximum number of minimization parameters (which affects the size of the internal one–dimensional MERLIN work arrays), and the precision (single or double) of floating point numbers. Although we could provide reasonable defaults, any attempt to change them, would become quite tedious, due to the size and complexity of the source code. Instead of hard–coding a default value in the source, we made use of an *installer macro*. An installer macro is a symbolic name that is referenced in the source as `$(`*name*`)`. Therefore the distribution files contain statements like:

```
IMPLICIT $(TYPE) (A-H,O-Z)
PARAMETER ( MXV = $(MXV) )
PARAMETER ( MXT = $(MXT) )
```

In the above statements the installer macros `$(TYPE)`, `$(MXV)` and `$(MXT)` must be substituted with actual values before the programs can be compiled. Hence, the term *installation* refers to the production of a valid Fortran source code from the distribution files. After installation is complete the above statements might look like:

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
PARAMETER ( MXV = 200 )
PARAMETER ( MXT = 400 )
```

A complete list of all installer macros used in the MERLIN–MCL source code is given in §2.2.3.

2.2.2 Running the installer program

To ease the installation process, we have developed a small installer program, written in Fortran–77. Its main purpose is to replace all instances of the installer macros in the MERLIN–MCL distribution files with actual values, and output a valid Fortran source, ready for compilation. The actual values to be substituted are supplied interactively by the user. Note that the source code for both, MERLIN and MCL must be processed by the installer.

Compiling and running the installer program is straightforward. For example on a Unix system:

```
f77 -O -o install install.f
./install
```

```
TYPE    'DOUBLE PRECISION'
MXV     200
MXT     200
CONFIG  CONFIG
FC      ''
MCLMEM  50000
MCLBUF  50000
MXPLUG  50
```

Figure 2.1: Sample installer definitions file.

Installation is then carried out in three steps:

1. The installer will interactively ask you for the values of the most important installer macros. Then it will record these values in the *installer definitions file* (with the default name `DEFS`) and use them in the next step.

   The format of the installer definitions file is simple: each line contains the name of an installer macro and the actual value to be substituted in the source code. Blank lines and lines starting with the `%` character (comments) are ignored. Items containing space or tab characters must be enclosed in a pair of quotes. A single quote is produced by the sequence `\'`. A sample definitions file is shown in figure 1.

   The installer will allow you to prepare a definitions file yourself, instead of creating one interactively. In this case you must include in the installer definitions file a value for *all* installer macros listed in §2.2.3.

2. The installer will read the installer definitions file (either the one created by the installer itself, or the one prepared by the user) and ask the names of the distribution files. Then it will read the contents of the distribution files, substitute the actual values in place of the installer macros, and output the full Fortran–77 source, ready for compilation.

3. The last step is compilation of the package. The installer cannot do this for you since invoking a compiler is a system dependent task. Compilation of the processed MERLIN–MCL source is quite straightforward. MCL is an autonomous program and must be compiled alone. For example on a Unix system:

   ```
   f77 -O mcl.f -o mcl
   ```

   MERLIN should be compiled and linked with the appropriate user written routines. The distribution provides a sample:

Hardware	*Operating System*	*Compiler*
SUN Ultra–2	SunOS 5.5.1	f77 4.0
SUN SparcClassic	SunOS 4.1.3C	f77 1.4
SGI Power Challenge–M	IRIX 6.2	f77 7.0
Control Data 4680	EP/IX 1.4.3	f77 2.20
PC-486	MS–Windows 95	MS Fortran PowerStation 4.0
PC-486	Linux 2.0.18	f2c 19950110, gcc 2.7.0
Macintosh Performa 6320	MacOS 7.5.1	MPW 3.3, LS Fortran 3.3
Macintosh LC475	MacOS 7.1	Absoft Mac–Fortran 2.4

Table 2.1: Some systems under which MERLIN was successfully installed.

```
f77 -O -c merlin.f
f77 -O -c sample.f
f77 -o merlin merlin.o sample.o
```

Make sure the panel description file `PDESC` is present before attempting to run the MERLIN or MCL executables.

Using the above procedure, we were able to successfully install MERLIN in a number of different systems in single and double precision. Some of them are listed in table 2.1.

2.2.3 The complete list of installer macros

All of the following installer macros are used in the MERLIN source code. In addition, `TYPE`, `MXPLUG` and `FC` are used in the MCL source as well.

- `TYPE` Specifies the Fortran type for floating point variables. It should be set to either `REAL` or `'DOUBLE PRECISION'`. The installer will ask for the appropriate setting. Note that both MERLIN and MCL must be installed in the same precision in order for the programs to correctly cooperate.

- `MXV` The maximum number of optimization parameters. The setting of `MXV` determines the size of a few one–dimensional arrays used as work space.

- `MXT` The maximum number of terms when the objective function is coded as a sum of squared terms. The setting of `MXT` determines the size of a few one–dimensional arrays used as work space. If you do not intend to code the objective function as as sum–of–squares, set `MXT` to 1.

- `CONFIG` Name of the MERLIN–MCL configuration file. Configuration is described in §2.3 and in chapter 8. The default name substituted by the installer is `CONFIG`.

- `FC` Some systems use the first character of an output line as a terminal control character. In order to use a terminal control character, set `FC` to `1X,`. The installer always sets `FC` to `''` (no control character).

- `MCLMEM` The size of an array, available as run–time memory for an MCL program. The array is of type `REAL` or `DOUBLE PRECISION`, depending on the setting of `TYPE`. If you do not intend to run MCL programs set `MCLMEM` to 1. If during the installation process you specify that you intend to run MCL programs, the default value 50000 is substituted; otherwise `MCLMEM` is set to 1.

- `MCLBUF` The length of a character variable used as the MCL program buffer. This buffer is supposed to hold the entire MERLIN object code during execution of an MCL program. If you do not intend to run MCL programs set `MCLBUF` to 1 (MCL programs will still run; at a slower pace however). If during the installation process you specify that you intend to run MCL programs, the default value 50000 is substituted; otherwise `MCLBUF` is set to 1.

- `MXPLUG` The maximum number of plug–in modules. The installer always sets `MXPLUG` to 50. You should not change the default value, unless you are prepared to make the necessary modifications in the source code as well (add empty plug–ins, `CALL` and `GOTO` statements etc.).

2.3 Configuration

Although we made every effort to ensure the code is truly portable, there is a small number of system dependent options, that must be customized before MERLIN or MCL begins execution. The most important ones are the input–output unit numbers, and an approximation of the largest and smallest positive (distinguishable from zero) floating point numbers your machine accepts. The internal MERLIN defaults for these parameters are set in the source code to 5, 6, 10^{30} and 10^{-30} correspondingly; adequate for most systems in single precision. Any change to these parameters must be recorded in a configuration file, named `CONFIG`[1], that is read by MERLIN and MCL as well. In fact the purpose of the configuration file is twofold: Machine dependent parameters can be specified in `CONFIG`. On the other hand, the configuration file is used to alter the defaults, until one reaches the MERLIN prompt.

Each line of the configuration file contains a configuration directive:

[1]The default name can be changed during installation, using the `CONFIG` installer macro.

```
% This line is a comment since it starts with the % character.
% The largest floating point number.
BIGGER 1.D300
% The smallest positive, distinguishable from zero, floating point number.
SMALLER 1.D-300
% This system uses some peculiar input–output units.
INPUT_UNIT 9
OUTPUT_UNIT 9
```

Figure 2.2: Sample configuration file.

directive $parameter_1$ $parameter_2$ $\ldots parameter_k$

As usual, to embed spaces in a parameter, you must enclose it in a pair of singe quotes. Lines starting with the percent character (comments) and blank lines are ignored. The contents of a typical configuration file are shown in figure 2.2. The directives `INPUT_UNIT` and `OUTPUT_UNIT` refer to the input and output unit correspondingly, while directives `BIGGER` and `SMALLER` refer to the largest and smallest positive (distinguishable from zero) floating point numbers. There more directives that change the defaults and control the behavior of MERLIN during startup; they are explained in detail in chapter 8.

Chapter 3

User written programs

MERLIN offers an optimization environment, supplies several utility as well as minimization tools, so that the user can choose whatever is convenient and effective each time. The programs the user may have to write are the following:

1. The main program.
2. The objective function in a general or in a sum–of–squares form.
3. The gradient vector (optional).
4. The Jacobian matrix (optional).
5. The Hessian matrix (optional).

In what follows we give sample programs coded in single precision. These should be used with a single precision installation. If double precision is needed, which is quite often the case, one should use a double precision installation, and code these programs in double precision.

3.1 The main program

The role of the main program is to initiate MERLIN's execution. The user may write his own main program or may use the sample provided below modified appropriately to meet his needs. To invoke MERLIN one should make a call to the subprogram:

```
SUBROUTINE MERLIN ( N, M, VERSIM, MAXW, IQUIT )
```

`N` (input) is the dimensionality of the problem.

`M`	(input)	is the number of the squared terms (useful for the sum of squares form)
`VERSIM`	(workspace)	is a work–space array dimensioned as: `VERSIM(MAXW)`
`MAXW`	(input)	$\max\{N(N+11), NM\}$
`IQUIT`	(output)	is an output flag that is either specified by the user at run–time, or set by the MERLIN Operating System to indicate an error condition. Possible values are:

- 0 A `RETURN` command was issued by the user.
- ≥ 0 A `QUIT` command was issued, with the specified value for `IQUIT`.
- -1 There is not enough one–dimensional storage for this problem. You must redimension (by running the installer) and recompile MERLIN.
- -2 MERLIN cannot allocate the initial unit numbers for the configuration file.
- -3 The output file or unit cannot be opened. Moreover, some errors occurred while parsing the configuration file.
- -4 An error occurred while parsing the configuration file. The error is related to directives that set the output unit or file.
- -5 The output file or unit cannot be opened. The configuration file has been successfully parsed.
- -6 An error occurred while parsing the configuration file. The error is not related to the output file or unit which has been successfully opened.
- -7 An error occurred while parsing the panel description file.
- -8 The input unit cannot be opened.
- -10 An End–of–File condition was encountered in the input file or unit.
- -20 A run time error, other than End–of–File has occurred (incorrect command, MCL runtime error, etc.). This value will be returned only if the `BATCH` mode was set.

A sample main program is listed in figure 3.1.

3.2 The objective function. General form

This must be written as a function subprogram:

```
      PROGRAM MASTER
C
C  Maximum number of variables.
      PARAMETER ( MXV = 200 )
C  Maximum number of terms.
      PARAMETER ( MXT = 200 )
C
C  The 2-dimensional storage required by some Merlin commands depends
C  on the maximum number of parameters (MXV) and the maximum number of
C  terms (MXT).
C  If MXV+11 > MXT one needs MAX1 storage locations:
      PARAMETER ( MAX1 = MXV**2+11*MXV )
C  Otherwise:
      PARAMETER ( MAX2 = MXV*MXT )
C  For this example we chose MAX1 (actually MAX2 would be ok too).
      PARAMETER ( MAXW = MAX1 )
C  Alternatively if you don't have that much storage available,
C  set MAXW to any positive value. Commands that need more storage
C  will not be executed by Merlin.
      DIMENSION VERSIM(MAXW)
C
      WRITE (*,*) 'Enter number of variables, number of squared terms:'
      READ (*,*) N, M
      CALL MERLIN(N,M,VERSIM,MAXW,IQUIT)
C
      END
```

Figure 3.1: A sample main program.

```
FUNCTION FUNMIN ( X, N )
DIMENSION X(N)
X1 = X(1)
X2 = X(2)
FUNMIN = (10*(X2-X1**2))**2 + (1-X1)**2
END
```

Figure 3.2: A sample FUNCTION FUNMIN.

```
FUNCTION FUNMIN ( X, N )
DIMENSION X(N)
```

X (input) is an array holding the values of the parameters x_i.

N (input) is the dimensionality of the problem.

FUNMIN (output) upon return assumes the value of the objective function $f(\boldsymbol{x})$.

An example for the Rosenbrock function $f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ is shown in figure 3.2.

3.3 The objective function. Sum of squares form

This must be written as a subroutine subprogram:

```
SUBROUTINE SUBSUM ( M, N, X, F )
DIMENSION X(N), F(M)
```

M (input) is the number of the squared terms.

N (input) is the dimensionality of the problem.

X (input) is an array holding the values of the parameters x_i.

F (output) is an array holding the values of the terms $f_i(\boldsymbol{x})$.

An example is given in figure 3.3 again for the Rosenbrock function with $f_1 = 10(x_2 - x_1^2)$ and $f_2 = 1 - x_2$.

```
      SUBROUTINE SUBSUM ( M, N, X, F )
      DIMENSION X(N), F(M)
      X1 = X(1)
      X2 = X(2)
      F(1) = 10*(X2-X1**2)
      F(2) = 1-X1
      END
```

Figure 3.3: A sample `SUBROUTINE SUBSUM`.

```
      SUBROUTINE GRANAL ( N, X, GRAD )
      DIMENSION X(N), GRAD(N)
      X1 = X(1)
      X2 = X(2)
      T = X2-X1**2
      GRAD(1) = -400*X1*T - 2*(1-X1)
      GRAD(2) = 200*T
      END
```

Figure 3.4: A sample `SUBROUTINE GRANAL`.

3.4 The gradient vector

This must be written as a subroutine subprogram:

```
SUBROUTINE GRANAL ( N, X, GRAD )
DIMENSION X(N), GRAD(N)
```

`N`	(input)	is the dimensionality of the problem.
`X`	(input)	is an array holding the values of the parameters x_i.
`GRAD`	(output)	is an array holding the values of the gradient components $\frac{\partial f}{\partial x_i}$.

An example for the Rosenbrock function is given in figure 3.4. Note that in earlier versions of MERLIN this routine had a different calling sequence.

3.5 The Jacobian matrix

This must be written as a subroutine subprogram:

```
      SUBROUTINE JANAL ( M, N, X, FJ, LD )
      DIMENSION X(N), FJ(LD,N)
      X1 = X(1)
      X2 = X(2)
      FJ(1,1) = -20*X1
      FJ(1,2) = 10
      FJ(2,1) = -1
      FJ(2,2) = 0
      END
```

Figure 3.5: A sample `SUBROUTINE JANAL`.

```
      SUBROUTINE JANAL ( M, N, X, FJ, LD )
      DIMENSION X(N), FJ(LD,N)
```

- `M` (input) is the number of the squared terms.
- `N` (input) is the dimensionality of the problem.
- `X` (input) is an array holding the values of the variables x_i.
- `FJ` (output) is an array holding the values: `FJ(I,J)` $= \frac{\partial f_i}{\partial x_j}$.
- `LD` (input) is the leading dimension of the matrix `FJ` used by MERLIN to store the Jacobian.

An example for the Jacobian of the Rosenbrock test–function is shown in figure 3.5.

3.6 The Hessian matrix

This must be written as a subroutine subprogram:

```
      SUBROUTINE HANAL ( H, LD, N, X )
      DIMENSION H(LD,N), X(N)
```

- `H` (output) is an array holding the values of the Hessian elements $G_{ij}(\boldsymbol{x})$.
- `LD` (input) is the leading dimension of the matrix `H` used by MERLIN to store the Hessian.
- `N` (input) is the dimensionality of the problem.
- `X` (input) is an array holding the values of the parameters x_i.

```
      SUBROUTINE HANAL ( H, LD, N, X )
C  Note: We only have to fill the lower triangular (including the
C  diagonal) part of matrix H.
      DIMENSION H(LD,N), X(N)
      X1 = X(1)
      X2 = X(2)
      H(1,1) = 1200*X1**2 - 400*X2 + 2
      H(2,1) = -400*X1
      H(2,2) = 200
      END
```

Figure 3.6: A sample SUBROUTINE HANAL.

Note that *only the lower triangular part of* H *must be filled in.* The rest of the Hessian matrix is completed by MERLIN, using symmetry. An example for the Hessian of the Rosenbrock function is shown in figure 3.6.

3.7 Important note

The user must always construct one of the FUNCTION FUNMIN, or the SUBROUTINE SUBSUM subprograms. However a dummy routine must be provided for the one that is left out, since many linkers will not create the executable file otherwise. We list below examples for the dummy subprograms:

```
FUNCTION FUNMIN ( X, N )
DIMENSION X(N)
END

SUBROUTINE SUBSUM ( M, N, X, F )
DIMENSION X(N), F(M)
END
```

The same action must be taken when the user does not wish to code the SUBROUTINE GRANAL, the SUBROUTINE JANAL and the SUBROUTINE HANAL subprograms. The dummy routines should read as:

```
SUBROUTINE GRANAL ( N, X, GRAD )
DIMENSION X(N), GRAD(N)
END

SUBROUTINE JANAL ( M, N, X, FJ, LD )
DIMENSION X(N), FJ(LD,N)
```

```
END

SUBROUTINE HANAL ( H, LD, N, X )
DIMENSION H(LD,N), X(N)
END
```

Chapter 4

The MERLIN Operating System

MERLIN provides an operating system, in order to render the optimization process efficient, flexible and programmable. MERLIN has a command interpreter as a front end, that accepts the input commands and instructs the MERLIN's kernel to take appropriate action. Like most operating systems, the MERLIN Operating System (MOS) supports argumentary command packages called *macros* as well as command aliasing. In addition, there is the Merlin Control Language (MCL) through which one can device intelligent minimization strategies. The MCL compiler is a separate software package built specifically for this purpose.

This chapter presents the most important MERLIN features. A detailed description of all MERLIN commands is given in chapter 6.

4.1 Commands

MOS offers a repertoire of commands. These can be classified into categories according to their action. In order to present the general idea we will mention the most essential categories.

- Commands that manipulate the attributes of the minimization parameters. The i^{th} minimization parameter, has the following attributes:
  - Its current value x_i.
  - A unique symbolic name, up to 10 characters long, that may be set by the user.
  - An indication whether this parameter is fixed. This is referred to as the fix status of the parameter. The minimization routines do not alter the value of fixed parameters.
  - An indication whether there exist lower and/or upper bounds for this parameter. This is referred to as the margin status of the parameter. The minimization routines make sure they never evaluate the function outside the allowed bounds.

  - The lower bound l_i, if it exists.
  - The upper bound u_i, if it exists.

- Commands that invoke one of the coded minimization algorithms.

- Commands that set the options for the various operation modes. These modes determine the overall behavior of MERLIN. For example there is the *printout* mode that determines the amount of output issued by MERLIN, the *gradient* mode that determines the manner in which the gradient components are estimated, the operation mode that can be set to either error tolerant, for interactive use, or strict, for batch processing, etc.

- Commands that issue information about the current state of both the optimization process and the system. For example command `SHORTDIS`, displays the attributes of the current minimization parameters, the corresponding value of the objective function and the number of function and gradient calls spent so far. Also the command `MODEDIS` displays the currently set options for the various modes.

- Commands that perform file manipulation operations. For instance the command `DELETE` that erases a file or the command `MEMO` that appends the current attributes of the minimization parameters to a file, etc.

- Commands that deal with the construction of macro packages, aliasing and other utilities.

4.2 Supplying input

Most MERLIN commands accept arguments that specify their action. Arguments are either mandatory or optional. The most general form of a command has the following structure:

command *mandatory_arguments optional_arguments*

Note that a command may or may not have either mandatory or optional arguments. One may proceed by entering only the command's name. In that case MERLIN will prompt only for the mandatory, if any, and not for the optional arguments, i.e. in this case the syntax is:

`command` ⟨cr⟩
mandatory_arguments

MOS first checks that the entered command is a valid one, in which case proceeds with a syntax check of the following data, otherwise issues a warning message and prompts for another command. Blank lines and comment lines do not cause warnings. A comment line is any line that its first non–blank character is the `%` symbol. Comment lines can be used for documentation purposes and are useful inside macros and in batch processing where an input file is prepared containing all the commands to be processed.

The input arguments are acquired using a mechanism called *the one line input.* The required input must be supplied by the user in a line of the form:

$item_1$ $item_2$ $item_3$ ... $item_k$

Items are separated by one or more spaces. The significance of each item varies according to the specific MERLIN command requesting the input. Commands that manipulate the minimization parameters use a scheme to specify which parameters are to be manipulated. This specification scheme uses:

1. Parameter indices.
   To operate on parameter x_i its index i is used. For example the entry: `FIX 1 5 7` fixes variables x_1, x_5 and x_7 to their current values.

2. Parameter names.
   A name corresponds to an index. It is just an auxiliary mnemonic alternative. Hence names and indices may be used invariably. Names can be set using the **`GODFATHER`** command.

3. Parameter ranges.
   A range is denoted as: $index_1 - index_2$ and specifies all parameters x_{index_1} through x_{index_2}. For example the entry: `FIX 1-5` fixes the parameters x_1 through x_5 to their current values. A range denoted as $index_1-$ corresponds to variables x_{index_1} through x_n. Similarly a range denoted as $-index_2$ corresponds to variables x_1 through x_{index_2}.

4. Shared parameter properties.
   One can specify a symbol that refers to some property of the minimization parameters. For example `SHORTDIS /F` will output the fixed parameters only, `SHORTDIS 1-10 25-30 /F` will display the parameters x_1 through x_{10}, x_{25} through x_{30}, and of all the fixed variables. The complete list of supported properties is shown in table 4.1.

5. In addition if the negation symbol `!` is set in front of an index specification, the meaning of the specification is reversed. For example a `!` in front of an index or a name excludes the corresponding parameter. Also a `!` in front of a range excludes the parameters in that range, and so on so forth. A `!` in front of an attribute specification excludes the parameters possessing this attribute. Note that no spaces nor tabs are allowed in between the negation operator `!` and the specification that follows. Example: **`GRADDIS !/F`**, **`LDEMARGIN !/M`**.

Panels are based on the one line input and are used to display and change the input parameters for some of the MERLIN commands. A panel displays in a tabular form all parameters that may be changed, their corresponding keywords, a short description of their function, their default values, and a list or range of allowed values. A typical panel is shown in figure 4.1. The panel prompts the user to change the current values, supplying an one line input, of the form:

Property	*Description*
`/F` or `/FIX`	Fixed variables
`/L` or `/LOOSE`	Loose variables
`/LM` or `/LEFT`	Variables with left margin
`/RM` or `/RIGHT`	Variables with right margin
`/M` or `/MARGIN`	Variables with both margins
`/FD` or `/FAST`	Variables whose derivative mode is set to `FAST`
`/QD` or `/QUAD`	Variables whose derivative mode is set to `QUAD`
`/ND` or `/NUMER`	Variables whose derivative mode is set to `NUMER`
`/AD` or `/ANAL`	Variables whose derivative mode is set to `ANAL`
`/N` or `/NAMED`	Named variables

Table 4.1: Shared parameter properties used in index specifications.

```
------------------------------------------------------------------------------
                                BFGS Panel
------------------------------------------------------------------------------
Ind Keyword    Description                    Value     Allowed values
 1) NOC        Number of calls                 300      Any int >= 1
 2) PRINT      Printout level                   1       {0,1,2}
 3) GTOL       G-convergence criterion        1.E-15    0. < real < 1.
 4) XTOL       X-convergence criterion        1.E-15    0. < real < 1.
 5) FTOL       F-convergence criterion          0.      0. <= real < 1.
 6) ITER       Iterations (-1 = Inf)           -1       Any int >= -1
 7) MAD        Automatic derivatives           off      {on,off}
 8) USEG       Use/Recalculate gradient         0       {1,0}
 9) USEH       Use/Recalculate Hessian          0       {1,0}
10) LS         Line search conditions         Weak      {Strong,Weak}
11) LSITER     Max. LS iterations              30       Any int >= 1
12) RHO        Line search rho               0.0001     0. <= real <= 1.
13) SIGMA      Line search sigma               0.9      0. <= real <= 1.
14) CANCEL     Cancel / Proceed                 1       {0,1}
------------------------------------------------------------------------------
Enter changes:
```

Figure 4.1: Typical MERLIN panel.

parameter$_1$ *value*$_1$ *parameter*$_2$ *value*$_2$ ... *parameter*$_k$ *value*$_k$

Parameters are of course specific to the command being used and may be entered using either their keyword, or their index in the panel. For example, for the `BFGS` panel shown above, any of the following two lines can be used invariably for input.

```
NOC 1000 PRINT 2
1 1000 2 2
```

An empty line signifies no changes. Parameters that are changed, are recorded by the panel and appear the next time the command is invoked. Panels also communicate information to an MCL program, upon completion of a command's execution; a subject covered in a later section. Each panel may be active (on) or inactive (off). Active panels appear and prompt the user for changes when the associated command is invoked. Inactive panels do not appear; their stored values are automatically used by the command. Panels are turned on or off with the commands `PANELON` and `PANELOFF` correspondingly. If used without an accompanying argument these commands turn on or off all MERLIN panels. Arguments may be used to alter the status of some of the panels. For example, the command:

```
PANELOFF BFGS DFP
```

turns off the panels associated with the commands `BFGS` and `DFP`. The status of all panels can be displayed with the `PSTATUS` command.

Panel commands can use a syntax involving two special characters the semicolon (`;`) and the exclamation (`!`) as directives in the following way.

panel_command keyword$_1$ *value*$_1$ *keyword*$_2$ *value*$_2$ ... `;`
panel_command keyword$_1$ *value*$_1$ *keyword*$_2$ *value*$_2$ ... `!`

In the first example *keyword*$_1$, *keyword*$_2$, ... assume the values *value*$_1$, *value*$_2$, ... The panel appears displaying these newly substituted values and prompts for additional input. In the second case *keyword*$_1$, *keyword*$_2$, ... assume the values *value*$_1$, *value*$_2$, ... which are substituted in the panel. The *panel_command* is not executed however, and no panel appears.

4.3 Output handling

MERLIN produces two kinds of output, the informative (normal) output, and the error output. Normal output is produced by almost every command, while error output results when some abnormal situation is encountered. The amount of output produced by MERLIN is controlled by the commands `FULLPRINT`, `HALFPRINT`, and `NOPRINT` that set the options for the printout mode. Note that some commands contain options (most notably the `PRINT` keyword in all minimization command panels) that allow further control on the output when the `FULLPRINT` option is in effect.

MERLIN output may be temporarily redirected from its normal destination (a terminal or a file) to any file using the `>` and `>>` symbols, followed by a filename. For example, the command:

```
SHORTDIS > result
```

redirects the output of the `SHORTDIS` command to a file named `result`, overwriting its previous contents. The symbol `>>` appends the output to the contents of an already existing file. Note that both the normal as well as the error output are redirected. The effect of output redirection can be facilitated for a number of subsequent commands, using a pair of `HIDEOUT` – `REVEAL` commands. The command:

```
HIDEOUT file
```

redirects all subsequent output to the specified file, until a `REVEAL` command is entered. The file is overwritten. To append to the specified file rather than overwriting it one must enter the following:

```
HIDEOUT file append
```

Nested pairs of `HIDEOUT` – `REVEAL` commands are allowed, as well as commands with `>` and `>>` redirection, inside a `HIDEOUT` – `REVEAL` block.

4.4 Getting help

Since MERLIN is designed primarily for interactive use, it is equipped with an easy to use help system. The user can request help on any MERLIN command. The help texts include the command syntax, a short description of the command, and pointers to other relevant information.

The help system is facilitated by the `HELP` command. Without any arguments it displays a help screen, instructing the user how to request further help. To get information about a specific command or topic, use `HELP` with an argument, for example: `HELP BFGS`. To get an alphabetic list of all MERLIN commands, enter the command `LIST`.

Help is also available on the usage and meaning of the various keywords in the panel commands. When prompted by a panel for changes, enter the name of any keyword followed by a question mark, for example `PRINT ?`.

The MERLIN help system can be easily extended, if one wishes to add his own commands or topics, or even enlarge the description of the already existing texts. The procedure is described in detail in chapter 7.

4.5 Minimization commands

The minimization algorithms are invoked using the commands `BFGS`, `DFP`, `TRUST`, `TOLMIN`, `CONGRA`, `ROLL`, `SIMPLEX`, `LEVE` and `AUTO`. They all use the panel mechanism to obtain their input parameters, and set the termination criteria. All minimization commands use a few keywords with the same meaning:

`NOC` is an approximate upper bound to the allowed number of evaluations of the objective function.

`PRINT` controls the amount of output by the minimization algorithm. Allowed values are:

- 0 suppresses all output.
- 1 reports each lower function value as it is discovered.
- 2 reports the function value as well as the values of the minimization parameters.

`CANCEL` controls whether execution of the minimization algorithm will proceed. Allowed values are:

- 0 causes the user changes to be recorded in the panel, the minimization algorithm will not be executed however.
- 1 causes minimization to proceed normally.

Upon termination all minimization commands report the number of function and gradient calls that were performed as well as the number of iterations of the method.

MERLIN can estimate derivatives of the objective function using finite differences. Three ways are implemented and correspond to the following modes of operation, listed in order of increasing accuracy and computational cost: `FAST`, `QUAD`, `NUMER`. The minimization methods that use derivatives have the so called `MAD` option in their panel, standing for MERLIN Automatic Derivatives. This option, when it is turned on, allows according to some criterion the automatic transition from a lower to a higher accuracy mode. This is done separately for each component of the gradient. This means that at a particular instant, some components will be calculated in the `FAST` mode, some others in the `QUAD` mode, etc. This is designed so as to economize calls to the objective function. The criterion for a transition is based on the absolute value of the gradient component. Two numbers (thresholds) are specified, one for the transition from `FAST` to `QUAD` and another for the transition from `QUAD` to `NUMER`. A derivative changes from `FAST` to `QUAD` if its absolute value is below the first threshold. Similarly the second threshold arranges for the `QUAD` to `NUMER` transition. The rational behind this criteria is that as the minimum is approached, the gradient components tend to become smaller and smaller in absolute value. The closer to the minimum the more accurate estimation of

MERLIN commands	*Description*
`HESSIAN DO C`	Calculate the Hessian
`HESSIAN DO D`	Decompose the Hessian to its Choleski factors
`BFGS NOC 1000 USEH 1`	Instruct `BFGS` it to use the existing Choleski factors

Table 4.2: Using an initial Hessian approximation.

the gradient is needed. The thresholds can be set via the `MAD` command and their values depend on the problem at hand.

MERLIN can also estimate the Hessian and the Jacobian numerically. For the Jacobian forward differences are used. The Hessian can always be calculated by using function values. However if the user has supplied his own code for the gradient MERLIN can estimate the Hessian using these gradient code. The appropriate formulas are given in chapter 6. The minimization commands `BFGS`, `DFP` and `TRUST` can take advantage of this capability, by using this rather well estimated Hessian initially, instead of the unit matrix. So one first calculates the Hessian and then instructs one of the above three commands to employ it as an initial approximation. This is facilitated by setting the `USEH` parameter in their panels equal to 1. However since the routines do not use the Hessian directly, but its Choleski factors, after calculating the Hessian, one must decompose it and then proceed with one of the above commands. The relevant input to achieve this is listed in table 4.2.

4.6 Macros

A macro is a collection of MERLIN commands and calls to other macros. Macros are identified by a symbolic name, and are stored either individually (each macro in a separate file) or collectively in a special multi–macro file, named `MACROF`[1]. A macro stored in a separate file, is named after the filename. Macros stored in file `MACROF`, have their names explicitly declared inside the file, along with the constituent commands. A sample `MACROF` is shown in table 4.3.

A macro is invoked using its name, prepended by a dot. For example, a macro named `test` would be invoked as

```
.test
```

When a macro is invoked MERLIN first searches the multi–macro file `MACROF` and then, if the macro cannot be found there, tests for a file with the same name. Macros can contain an arbitrary number of commands and calls to other macros, the only restriction being that macro recursion is not allowed. Macros can accept optional arguments that are designated as `[1]`, `[2]` etc. inside the

[1]The default name `MACROF` can be changed using the `MACRO_FILE` configuration directive.

File contents	*Description*
`*`	Macro separator
`.s`	Macro name
`SHORTDIS 1-5`	The first macro starts here
`STEPDIS 1-5`	
`CLEAR`	The first macro ends here
`*`	Macro separator
`.mini`	Macro name
`POINT 1- 4`	The second macro starts here
`ANAL`	
`BFGS NOC 2000 PRINT 0`	
`VALDIS`	
`GNORM`	
`CLEAR`	The second macro ends here

Table 4.3: Sample file `MACROF` with two macros.

Macro commands	*Description*
`ECHO The macro name is [0]`	Display the macro name
`ECHO You entered [#] arguments`	How many arguments did the user enter ?
`BFGS NOC [1] PRINT [2]`	Use the arguments in a `BFGS` call

Table 4.4: Sample macro that expects two arguments.

macro. When the macro is invoked, they are substituted by the actual arguments supplied by the user. The macro name itself is designated as `[0]`, while the number of arguments that are actually entered when the macro is invoked is designated as `[#]`. The sample macro `m` shown in table 4.4 expects two arguments and can be invoked as

```
.m 1000 0
```

or

```
.m 2000 1
```

Any text editor can be used to construct and modify a macro. MERLIN provides a simple facility to aid the construction of a macro. Command `MACRO` initiates an interactive process, that prompts for the macro name and its contents, and finally stores the macro in the specially formatted multi–macro file `MACROF`.

Macros can conveniently automate certain procedures. They are of limited use however, since they don't provide mechanisms for conditional execution of commands or transfer of control. One should use the MERLIN Control Language to implement complex tasks.

4.7 Confidence intervals

One of the most common uses of MERLIN is to fit a set of M data points (t_i, y_i) to a model function $y(t; \boldsymbol{x})$, that depends on the N unknown parameters $\boldsymbol{x}$. Each measurement y_i has its own uncertainty e_i. Assuming the measurement errors e_i are normally distributed around the true model $y(t)$, the most likely set of parameters $\boldsymbol{x}$ is determined by minimizing the chisquare function

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} \left(\frac{y_i - y(t_i; \boldsymbol{x})}{e_i} \right)^2 \tag{4.1}$$

The user is responsible for coding the objective function $f(\boldsymbol{x})$ as described in chapter 3, and invoke the MERLIN minimizers. After a minimum has been found, MERLIN is able to compute confidence intervals, based on the covariance matrix of the fit; the so–called parabolic errors. This approach assumes that either the model $y(t; \boldsymbol{x})$ is linear in its parameters, or the sample size M is large enough, so that the uncertainties in the fitted parameters $\boldsymbol{x}$ are inside a region in which the model could be replaced by a suitable linearized model.

Let ν be the number of parameters whose joint confidence region we wish to compute, and p the desired confidence limit. The covariance matrix $\boldsymbol{C}$ is defined as $\boldsymbol{C} = 2\, \boldsymbol{G}^{-1}$. The procedure involves two steps.

Initially using the `COVARIANCE` command, one must set the desired confidence limit p and calculate the covariance matrix of the fit. MERLIN calculates the second derivative matrix using finite differences, inverts it using Choleski decomposition, writes the resulting covariance matrix to a file for further work, and prints the standard errors $\sigma_i = \sqrt{C_{ii}}$.

Once the covariance matrix is calculated and stored, one must invoke the command `CONFIDENCE` with appropriate arguments, to calculate the joint confidence region for any number ν of parameters. For example, to calculate the confidence region for parameters 2 and 5, use the command `CONFIDENCE 2 5`. MERLIN reads the stored covariance matrix and forms a new $\nu \times \nu$ matrix $\widetilde{\boldsymbol{C}}$ using the intersections of the ν rows and columns of $\boldsymbol{C}$, corresponding to the parameters specified in the `CONFIDENCE` command. Using a simple bisection routine it solves the equation

$$Q(\nu/2, \Delta/2) = 1 - p \tag{4.2}$$

for Δ, and prints out the standard errors $\sigma_i = \sqrt{C_{ii}}$ and the corresponding confidence intervals $\delta x_i = \sqrt{\Delta}\ \sigma_i$. $Q(\nu/2, \Delta/2)$ is the incomplete Gamma function, and Δ is a value such that the probability of a chisquare variable with ν degrees of freedom, being less than Δ is p. Finally MERLIN inverts $\widetilde{\boldsymbol{C}}$, and prints the equation of the ellipsoid that defines the confidence region:

$$\Delta = (\boldsymbol{x}^* - \boldsymbol{x})^T \widetilde{\boldsymbol{C}}^{-1} (\boldsymbol{x}^* - \boldsymbol{x}) \tag{4.3}$$

where $\boldsymbol{x}^*$ is the current minimizer of the objective function. The stored covariance matrix can be used by successive `CONFIDENCE` commands, in order to calculate the confidence regions for any combinations of the parameters.

4.8 Command abbreviations

MOS supports command abbreviations. One needs to enter only an initial part of the command name that however identifies it uniquely. For example `SH` is long enough for `SHORTDIS`, `PO` for `POINT`, `BF` for `BFGS` etc. However `STE` is a common part of the three commands `STEP`, `STEPALL`, `STEPDIS` and hence unique identification is impossible. In such a case MOS lists all matching commands to remind the user of the several possibilities. Note also that commands can be entered invariably in either lower or upper case.

4.9 The MERLIN flags and how to use them

MERLIN maintains two arrays for flags. One is of floating point type, while the other is of character type. At run time the user may specify values for these flags. The default initial values are zeros for the numerical flags and blanks for the character ones. Flags can be used to further extend the control of the MOS inside the user's program. There is a *glue routine* called `SUBROUTINE GETFLA` (see chapter 9 on glue routines) that the user can call from his code and then decide what to do according to the flag's value. This is very convenient if for example at some stage one decides that wants to print the values of some variables or arrays used in his program. Also a flag can be used to implement a penalty factor in order to solve a constrained optimization problem. This version of MOS supports twenty numerical and ten character flags, each one being 30 characters long. An example is listed in figure 4.2.

4.10 Call counters

MERLIN has eight counters. Their current values appear in the output produced by the commands `SHORTDIS` and `VALDIS`. The four of them count the calls to:

1. The objective function (`FUNCTION FUNMIN` or `SUBROUTINE SUBSUM`).

```
      FUNCTION FUNMIN ( X, N )
      DIMENSION X(N)
         . . .
C  Call the glue routine that will return
C  the current value of the #1 flag.
      CALL GETFLA(1,FLAG)
         . . .
      A = expression
      B = another_expression
         . . .
      FUNMIN = A**2 + B**2
C  Print the values of A and B
C  if the user has set the #1 flag to three.
      IF (NINT(FLAG).EQ.3) WRITE (*,*) A, B
      END
```

Figure 4.2: Usage of the MERLIN flags.

2. The user supplied code for the gradient (`SUBROUTINE GRANAL`).
3. The user supplied code for the Jacobian (`SUBROUTINE JANAL`) .
4. The user supplied code for the Hessian (`SUBROUTINE HANAL`).

Counting starts at the beginning of the run and these counters cannot be reset (total counters). The remaining four counters (called partial counters) behave identically, with the exception that their values can be reset to zero by the user, via the `RESET` command. This can be quite convenient when a comparison among different methods or strategies is sought.

4.11 MERLIN files

MERLIN uses ordinary disk files to store information during the minimization process. The smallest unit of information stored in a MERLIN file, is called a *record*. It consists of $N+1$ entries, one for each minimization parameter, containing its value, symbolic name, fix status and lower and upper bound. The last entry contains the corresponding value of the objective function.

Two types of files are handled by MERLIN: *Text* files, where all the information is stored as human–readable ASCII characters. They are implemented in Fortran using formatted sequential files. Although updating the contents of a text file can be quite slow, the file is portable across different computer platforms. On the other hand, there are *binary* files, where information is stored in machine–readable form. They are implemented in Fortran using unformatted direct access files. Updating the contents of such a file is fast, even for large files. The file however cannot be transfered in a different computer system. Text is the default type for all file operations.

Chapter 5

Curve fitting. A complete example

Suppose we have the points $(t_i, y_i), i = 1, 2, \ldots, M$ and we want to construct an appropriate function $p(x)$ such that $y_i \approx p(t_i)$, $\forall\ i = 1, 2, \ldots, M$. We model the function $p(t) = p(t, a_1, a_2, \ldots, a_n)$ with parameters $a_i, i = 1, 2, \ldots, N$ which will be varied so as to achieve the above goal. Equivalently we may claim that this corresponds to minimizing the following objective function with respect to the a_i's.

$$F(a_1, a_2, \ldots, a_N) = \sum_{i=1}^{M} [y_i - p(t_i, a_1, a_2, \ldots, a_N)]^2 \equiv \sum_{i=1}^{M} f_i^2(a_1, a_2, \ldots, a_N)$$

To refer to a specific example consider the 20 points shown in table 5.1 and try to fit them with

$$p(t, a_1, a_2, a_3, a_4) = a_1 + a_2 t + a_3 t^2 + a_4 t^3$$

The Jacobian, the gradient and the Hessian are given by:

$$J_{ij} = -t_i^{j-1}$$

$$g_j = -2 \sum_{i=1}^{m} f_i t_i^{j-1}$$

$$G_{jk} = 2 \sum_{i=1}^{m} t_i^{k+j-2}$$

The Fortran–77 code for the subroutines `SUBSUM`, `JANAL`, `GRANAL` and `HANAL` is shown in figures 5.1, 5.2, 5.3 and 5.4 correspondingly. We assume that the data points reside in a file named `datafile` with contents as shown in table 5.1. A complete MERLIN session for this example is shown in table 5.2.

i	t_i	y_i	i	t_i	y_i
1	0.2	3.69619	11	2.2	8.37241
2	0.4	3.57096	12	2.4	9.43215
3	0.6	3.60643	13	2.6	10.58280
4	0.8	3.78799	14	2.8	11.82240
5	1.0	4.10364	15	3.0	13.14940
6	1.2	4.54358	16	3.2	14.56230
7	1.4	5.09979	17	3.4	16.06010
8	1.6	5.76569	18	3.6	17.64200
9	1.8	6.53590	19	3.8	19.30710
10	2.0	7.40601	20	4.0	21.05490

Table 5.1: Data points to be fitted for the sample curve fit.

```
      SUBROUTINE SUBSUM ( M, N, X, F )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      SAVE
      PARAMETER ( NDATA = 20 )
      DIMENSION X(N), F(M)
      COMMON /XYDATA/ XDATA(NDATA), YDATA(NDATA)
      DATA ITRIC / 0 /
*
*  The following block-if is executed only once i.e.,
*  when the subroutine is called for the first time.
*
      IF (ITRIC.EQ.0) THEN
         ITRIC = 1
         OPEN (1,FILE='datafile')
         DO 1 I=1,M
 1          READ (1,*) XDATA(I), YDATA(I)
         CLOSE (1)
      END IF
*
      DO 2 I=1,M
         XD = XDATA(I)
         YMODEL = X(1)+X(2)*XD+X(3)*XD**2+X(4)*XD**3
         F(I) = YDATA(I)-YMODEL
 2    CONTINUE
      END
```

Figure 5.1: The objective function for the sample curve fit.

```
      SUBROUTINE JANAL ( M, N, X, FJ, LD )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER ( NDATA = 20 )
      DIMENSION X(N), FJ(LD,N)
      COMMON /XYDATA/ XDATA(NDATA), YDATA(NDATA)

      DO 2 J=1,N
         DO 2 I=1,M
2           FJ(I,J) = -XDATA(I)**(J-1)
      END
```

Figure 5.2: The Jacobian matrix for the sample curve fit.

```
      SUBROUTINE GRANAL ( N, X, GRAD )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER ( NDATA = 20 )
      DIMENSION X(N), GRAD(N)
      COMMON /XYDATA/ XDATA(NDATA), YDATA(NDATA)

      DO 3 J=1,N
         GSUM = 0.
         DO 2 I=1,NDATA
            XD = XDATA(I)
            YMODEL = X(1)+X(2)*XD+X(3)*XD**2+X(4)*XD**3
            FI = YDATA(I)-YMODEL
            GSUM = GSUM-2*FI*XD**(J-1)
2        CONTINUE
         GRAD(J) = GSUM
3     CONTINUE
      END
```

Figure 5.3: First partial derivatives for the sample curve fit.

```
      SUBROUTINE HANAL ( H, LD, N, X )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER ( NDATA = 20 )
      COMMON /XYDATA/ XDATA(NDATA), YDATA(NDATA)
      DIMENSION X(N), H(LD,N)

      DO 1 I=1,N
         DO 1 J=1,I
            HSUM = 0
            DO 2 K=1,NDATA
2              HSUM =  HSUM + 2*XDATA(K)**(I+J-2)
            H(I,J) = HSUM
1     CONTINUE
      END
```

Figure 5.4: The Hessian matrix for the sample curve fit.

Table 5.2: A complete MERLIN session for the curve fit example.
User input appears underlined.

```
 Enter number of variables, number of squared terms:
4 20
--------------------------------------------------------------------------------
...........................
...........................                    M E R L I N - 3.0
...........................
...........................             D.G. Papageorgiou, I.E. Lagaris
...........................                   I.N. Demetropoulos
...........................                 University of Ioannina
...........................                      G R E E C E
...........................
...........................               Email: merlin@nrt.cs.uoi.gr
...........................             Web: http://nrt.cs.uoi.gr/merlin
...........................
--------------------------------------------------------------------------------
The Merlin help file "HELP" is present.
The panel description file "PDESC" is present.
Use the "help" command to obtain on-line information.
--------------------------------------------------------------------------------
Number of terms:     20
Number of variables: 4
Estimated machine's accuracy: 1.E-15
Merlin uses "SUBROUTINE SUBSUM" as the objective function.

 ....   W A R N I N G   ....
...  Initialize variables  ...

    /\/\/\/\/\/\     Merlin is at your command !!!
sos                                 Instruct MERLIN to use the sum-of-squares form
SOS

Functional form has been set to "sum-of-squares".
Merlin will be calling "subroutine subsum".

 ....   W A R N I N G   ....
...  Initialize variables  ...

    /\/\/\/\/\/\     Merlin is at your command !!!
point 1 22.9 2-3 1.1 4 -11.4                                Enter a starting point
POINT

    /\/\/\/\/\/\     Merlin is at your command !!!
janal                Instruct MERLIN to use the user supplied code for the Jacobian
JANAL

    /\/\/\/\/\/\     Merlin is at your command !!!
title 'Curve Fitting using a cubic polynomial'          Set a title for the problem
TITLE

Title is set to: "Curve fitting using a cubic polynomial"

    /\/\/\/\/\/\     Merlin is at your command !!!
```

continued on next page

continued from previous page

```
shortdis
SHORTDIS

Title: Curve fitting using a cubic polynomial
--------------------------------------------------------------------------------
Number of evaluations:    Function       Gradient       Hessian      Jacobian
Total:                           1              0             0             0
Since last reset:                1              0             0             0
--------------------------------------------------------------------------------
 Index  Name       Fix       Parameter value       Left margin   Right margin
    1)             .......   22.9000000000000
    2)             .......   1.10000000000000
    3)             .......   1.10000000000000
    4)             .......  -11.4000000000000
       Value .......     1623492.70661034

   /\/\/\/\/\/\     Merlin is at your command !!!
leve noc 100              Start the minimization using the Levenberg–Marquardt method
LEVE
   Iter: 1   Lower value:     1623492.70661034        Calls: 1 of 100
--------------------------------------------------------------------------------
   Iter: 2   Lower value:    9.484394200265691E-03    Calls: 2 of 100
--------------------------------------------------------------------------------
   Iter: 3   Lower value:    9.484394200265485E-03    Calls: 3 of 100
--------------------------------------------------------------------------------
                    LEVE: Further progress is not possible
--------------------------------------------------------------------------------
Function evaluations:   5
Jacobian evaluations:   3
Iterations:             3
--------------------------------------------------------------------------------


   /\/\/\/\/\/\     Merlin is at your command !!!
shortdis
SHORTDIS

Title: Curve fitting using a cubic polynomial
--------------------------------------------------------------------------------
Number of evaluations:    Function       Gradient       Hessian      Jacobian
Total:                           6              0             0             3
Since last reset:                6              0             0             3
--------------------------------------------------------------------------------
 Index  Name       Fix       Parameter value       Left margin   Right margin
    1)             .......   3.85673558101135
    2)             .......  -1.37311723433982
    3)             .......   1.71949888365997
    4)             ....... -7.585181298786750E-02
       Value .......    9.484394200265485E-03

   /\/\/\/\/\/\     Merlin is at your command !!!
stop                                                         We are done. Good bye
STOP
---------------------------- Merlin run has ended ----------------------------
```

Chapter 6

Command description

6.1 Command classification

There are several ways in which the commands can be grouped together. In presenting the commands a grouping based on the context relevance is used. However a grouping according to the syntax format is also convenient for the command description. Hence some definitions are given below for the several syntax categories:

1. **Simple commands (SIC)**
   These commands need no arguments. They are called by issuing their names.
   Example: `RESET`

2. **Range commands (RAC)**
   These commands need only *range* specification arguments. Range specifications are described in §4.2.
   Syntax: *command_name* $range_1$ $range_2$ ...
   Example: `FIX 1-3 7-8 10`

3. **Range–value commands (RAVAC)**
   These commands need both *range* and *value* arguments.
   Syntax: *command_name* $range_1$ $value_1$ $range_2$ $value_2$ ...
   Example: `POINT 1-5 10 6 11.2 7-10 -3.1`

4. **Panel commands (PAC)**
   These commands need arguments as determined by the associated panel.
   Syntax: *command_name* key_1 $value_1$ key_2 $value_2$ ...
   Example: `ROLL NOC 1000 FAIL 8`

Ind	Keyword	Description	Value	Allowed values
1)	FILE	File to read from		Any string
2)	WHAT	What to initialize	x	{x,l,r,f,s,n}
3)	NPL	Numbers per line	0	Any int >= 0
4)	SKIP	Lines to skip	0	Any int >= 0
5)	FROM	Index to start from	1	Any int >= 1
6)	TO	Index to end to	2	Any int >= 1
7)	FORMAT	Format, eg: E20.10	*	Any string
8)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.1: The `INIT` panel.

5. **Unclassified commands (UNC)**
   There are a few commands that do not conform to the above types and do not deserve a separate classification.

6.2 Attribute related commands

6.2.1 `POINT`

Syntax classification: RAVAC

Purpose: Assigns values to minimization parameters.

Examples: `POINT 1-5 3.1 6 7 8- -1.2`
Assigns to variables x_1 through x_5 the value 3.1, to variable x_6 the value 7 and to variables x_8 through x_n the value -1.2.

6.2.2 `INIT`

Syntax classification: PAC

Purpose: Assigns names or values stored in a file to minimization parameters. In addition it can be used to assign values stored in a file to either the lower or upper bounds, to the fix–status of the variables, or to the search steps for the `ROLL` method. The associated panel is listed in figure 6.1. The panel parameters are as follows:

`FILE` The file holding the appropriate data.

`WHAT` Specifies what to initialize. Allowed values are:

- `X` Read in the current point.
- `L` Read in the left margins.
- `R` Read in the right margins.
- `F` Read in the fix statuses (1 for a free variable, 0 for a fixed one).
- `S` Read in the search steps used by the `ROLL` command.
- `N` Read in the symbolic names of the parameters.

`NPL` Specifies how many numbers will be read from each line of the file. This does not apply to symbolic names, which are always read one per line. Setting `NPL` to 0, lets the system decide for the actual numbers per line.

`SKIP` Determines an initial number of lines to be skipped before actually reading initialization data.

`FROM` Specifies the index to start the initialization.

`TO` Specifies the index to end the initialization.

`FORMAT` This is the format to be used when reading data from the file. You may specify any valid Fortran format, or `*` to use a free format. Note that the format must be consistent with the setting of `NPL`, containing the appropriate number of format specifiers. Symbolic names are always read one per line using an `A` format.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.2.3 PICK

Syntax classification: PAC

Purpose: Assigns values for all the parameter attributes by reading a specially formatted file. Such files may be created by other MERLIN commands (`MEMO`, `DISCARD`, `BACKUP`). The associated panel is listed in figure 6.2. The panel parameters are as follows:

`FILE` The file to pick from.

`TYPE` The file type. Allowed values are:

- `TEXT` for a MERLIN text file. Information is stored in human–readable, ASCII form. Slower but portable.

Ind	Keyword	Description	Value	Allowed values
1)	FILE	File to pick from		Any string
2)	TYPE	File type	Text	{Text,Bin}
3)	REC	Record to pick	1	Any int >= 0
4)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.2: The `PICK` panel.

- `BIN` for a MERLIN binary file. Information is stored in binary, machine–dependent form. Faster, but less portable.

`REC` The record to pick from the specified file.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.2.4 `LMARGIN, RMARGIN`

Syntax classification: RAVAC

Purpose: Set the lower and the upper bound correspondingly.

Examples: To restrict x_5 through x_8 in $[-3., 5.]$ and x_{10} in $[0, \infty)$ use the following:

```
LMARGIN 5-8 -3 10 0
RMARGIN 5-8 5
```

6.2.5 `LDEMARGIN, RDEMARGIN`

Syntax classification: RAC

Purpose: Clear the restrictions of the lower and upper bounds correspondingly.

Examples: To clear the lower bounds of parameters x_1 through x_3 use:

```
LDEMARGIN 1-3
```

6.2.6 `FIX`

Syntax classification: RAC

Purpose: Fixes specified parameters to their current values.

Examples: To fix parameters $x_1 - x_3, x_6$ and x_{10} use:

```
FIX 1-3 6 10
```

6.2.7 FIXALL

Syntax classification: SIC

Purpose: Fixes all parameters to their current values. This is used when one wants to let only a few variables free and it seems convenient to fix all variables and then loose a few.

Equivalent to: `FIX 1-`

6.2.8 LOOSE

Syntax classification: RAC

Purpose: Frees fixed variables.

Examples: To loose parameters $x_1 - x_3, x_6$ and x_{10} use:

```
LOOSE 1-3 6 10
```

6.2.9 LOOSALL

Syntax classification: SIC

Purpose: Frees all variables.

Equivalent to: `LOOSE 1-`

6.2.10 GODFATHER

Syntax classification: RAVAC

Purpose: Assigns symbolic names to the minimization parameters. Each name must be up to 10 characters long and is case insensitive.

Examples: To assign the name `sigma` to x_1 and the name `rho` to x_2 use the following:

```
GODFATHER 1 sigma 2 rho
```

6.2.11 NONAME

Syntax classification: RAC

Purpose: Clears parameter names.

Examples: To clear the names of x_1 to x_5 and that of x_8 use the following:
`NONAME 1-5 8`

6.2.12 SHORTDIS

Syntax classification: RAC

Purpose: Displays the current values of the minimization parameters and their attributes. In addition `SHORTDIS` displays the title of the run (if a title has been set, via the `TITLE` command), and the total and partial call counters for the objective function, the gradient, the Jacobian and the Hessian. The value of the objective function is also displayed at the end. Issuing simply `SHORTDIS` is equivalent to: `SHORTDIS 1-`

Examples: `SHORTDIS 3-5 7` displays the attributes of the variables x_3 through x_5 and that of x_7. To display the attributes of the fixed variables use: `SHORTDIS /f`.

6.2.13 VALDIS

Syntax classification: SIC

Purpose: Displays title and call counters (as the `SHORTDIS` command) and the value of the objective function. It does not display the parameter attributes.

Equivalent to: `SHORTDIS !1-`

6.2.14 TERMDIS

Syntax classification: RAC

Purpose: Displays the values of the M terms $f_i(\boldsymbol{x})$ in case the objective function has been coded as a sum of squares.

Examples: `TERMDIS 1-5` will display the values of the terms $f_1(\boldsymbol{x})$ through $f_5(\boldsymbol{x})$.

6.2.15 TITLE

Syntax classification: UNC

Syntax: `TITLE` *title_text*

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	FAIL	Failures allowed	4	Any int >= 1
4)	SFACTOR	Step factor	3.	Any real > 1.
5)	FTOL	Min. relative drop	0.	0. <= real < 1.
6)	LSNOC	Line search calls	30	Any int >= 1
7)	LSTOL	Line search tolerance	0.001	0. <= real < 1.
8)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.3: The `ROLL` panel.

Purpose: Sets a short title for the run. The title appears on top of the output produced by the `SHORTDIS` command.

Examples: `TITLE Rosenbrock Test Function`

6.2.16 `RESET`

Syntax classification: SIC

Purpose: Resets to zero the partial counters for the calls to the objective function, to the user supplied code for the gradient (`SUBROUTINE GRANAL`), the Jacobian (`SUBROUTINE JANAL`) and the Hessian (`SUBROUTINE HANAL`).

6.3 Minimization related commands

6.3.1 `ROLL`

Syntax classification: PAC

Purpose: Invokes the Roll minimization algorithm. The associated panel is listed in figure 6.3. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`PRINT` Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.

- 2 Display function and parameter values as well.

`FAIL` The number of successive cycle–failures allowed.

`SFACTOR` The step enhancement factor.

`FTOL` The minimum acceptable function drop per cycle.

`LSNOC` The maximum number of function calls that are allowed in the line search.

`LSTOL` The line search termination criterion. The search is terminated when the endpoints that bracket the minimum have a relative distance less than `LSTOL`.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

`FCALLS` Number of function evaluations that were performed.

`ITERDONE` Number of Roll iterations that were performed.

`INFO` Result code. Possible values are:

- 1 Target value has been reached.
- 2 All function evaluations have been used.
- 3 Too many cycle failures.

6.3.2 `SIMPLEX`

Syntax classification: PAC

Purpose: Invokes the Simplex minimization algorithm. The associated panel is listed in figure 6.4. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`PRINT` Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	500	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	USEV	Use/Recalculate vertices	0	{1,0}
4)	INIT	Initialization (Disp/LS)	1	{1,2}
5)	DISP	Displacement percent	0.1	Any real > 0.
6)	LSNOC	Line search calls	50	Any int >= 1
7)	LSTOL	Line search tolerance	0.01	0. <= real < 1.
8)	FTOL	Simplex tolerance	0.	0. <= real < 1.
9)	XTOL	X-convergence criterion	1.E-15	0. <= real < 1.
10)	ITER	Iterations (-1 = Inf)	-1	Any int >= -1
11)	ALPHA	Reflection factor	1.	Any real > 0.
12)	BETA	Contraction factor	0.5	0. < real < 1.
13)	GAMMA	Expansion factor	2.	Any real > 1.
14)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.4: The `SIMPLEX` panel.

- 2 Display function and parameter values as well.

`USEV` Specifies how `SIMPLEX` should obtain its initial vertices. Allowed values are:

- 0 Initialize the simplex vertices, according to the setting of `INIT`.
- 1 Use the vertices from a previous `SIMPLEX` invocation.

`INIT` Specifies the method of initializing the simplex vertices when `USEV` is set to 0. Allowed values are:

- 1 Use a displacement along each direction. The relative distance is specified by `DISP`.
- 2 Perform a line search along each direction and use the resulting minimum as a vertex.

`DISP` Specifies the relative distance along each direction when `INIT 1` is selected. More specifically, vertex i is set to the current point with the i^{th} parameter being:

$$x_i + \texttt{DISP} * |x_i|$$

`LSNOC` The maximum number of function calls that are allowed in the line search.

`LSTOL` The line search termination criterion. The search is terminated when the endpoints that bracket the minimum have a relative distance less than `LSTOL`.

`FTOL` The standard simplex termination criterion. `SIMPLEX` terminates when the standard deviation of the function values at the vertices is less than `FTOL`.

XTOL `SIMPLEX` terminates when either one of the following conditions hold:

- The standard deviation of every minimization parameter (with respect to all simplex vertices) is less than `XTOL`.
- The scaled distance between two successive iterates is less than `XTOL`.

ITER The number of Simplex iterations to perform. A value of -1 allows an unlimited number of iterations.

ALPHA The reflection factor.

BETA The contraction factor.

GAMMA The expansion factor.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

FCALLS Number of function evaluations that were performed.

ITERDONE Number of Simplex iterations that were performed.

INFO Result code. Possible values are:

- 1 The specified function accuracy has been achieved.
- 2 The specified number of iterations has been reached.
- 3 All function evaluations have been used.
- 4 Target value has been reached.
- 5 The X-convergence criterion is satisfied.
- 6 The simplex has become too small.
- 7 All variables are fixed.
- 8 Further progress is not possible.

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	GTOL	G-convergence criterion	1.E-15	0. < real < 1.
4)	XTOL	X-convergence criterion	1.E-15	0. < real < 1.
5)	FTOL	F-convergence criterion	0.	0. <= real < 1.
6)	ITER	Iterations (-1 = Inf)	-1	Any int >= -1
7)	MAD	Automatic derivatives	Off	{On,Off}
8)	USEG	Use/Recalculate gradient	0	{1,0}
9)	USEH	Use/Recalculate Hessian	0	{1,0}
10)	LS	Line search conditions	Weak	{Strong,Weak}
11)	LSITER	Max. LS iterations	30	Any int >= 1
12)	RHO	Line search rho	0.0001	0. <= real <= 1.
13)	SIGMA	Line search sigma	0.9	0. <= real <= 1.
14)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.5: The `BFGS` panel.

6.3.3 `BFGS`

Syntax classification: PAC

Purpose: Invokes the BFGS minimization algorithm using line search and Choleski decomposition for the Hessian. The associated panel is listed in figure 6.5. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`PRINT` Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.
- 2 Display function and parameter values as well.

`GTOL` The gradient termination criterion. The method terminates when the relative gradient falls below this value.

`XTOL` Termination criterion based on the values of the parameters. The method terminates when the relative change in the parameters in two successive iterations is less than `XTOL`.

`FTOL` Termination criterion based on function values. The method terminates when the relative function drop in two successive iterations is less than `FTOL`.

`ITER` The number of BFGS iterations to perform. A value of -1 allows an unlimited number of iterations.

`MAD` Specifies whether the MERLIN Automatic Derivatives are to be used when approximating the gradient vector. Allowed values are:

- `ON` The MERLIN Automatic Derivatives are to be used.
- `OFF` Use the current derivative mode.

`USEG` Determines how to obtain the gradient vector for the initial (current) point. Allowed values are:

- 0 Do not use the values in the gradient cache. Recalculate the whole gradient vector.
- 1 Instead of recalculating the gradient vector, use the values in the gradient cache.

`USEH` Determines the initial approximation to the Hessian matrix. Allowed values are:

- 0 The initial approximation is set to be the unit matrix.
- 1 The approximation from a previous `BFGS`, `DFP`, `TRUST` or `HESSIAN` command is used.

`LS` Specifies the conditions that terminate the line search. Allowed values are:

- `STRONG` The strong Wolfe–Powell conditions are used.
- `WEAK` The weak Wolfe–Powell conditions are used.

`LSITER` The maximum number of iterations that are allowed in the line search.

`RHO` The ρ parameter that defines an acceptable drop in function value.

`SIGMA` The σ parameter used in the termination criteria for the line search.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

`FCALLS` Number of function evaluations that were performed.

`GCALLS` Number of evaluations of the gradient vector.

`ITERDONE` Number of BFGS iterations that were performed.

`INFO` Result code. Possible values are:

- 1 Target value has been reached.

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	GTOL	G-convergence criterion	1.E-15	0. < real < 1.
4)	XTOL	X-convergence criterion	1.E-15	0. < real < 1.
5)	FTOL	F-convergence criterion	0.	0. <= real < 1.
6)	ITER	Iterations (-1 = Inf)	-1	Any int >= -1
7)	MAD	Automatic derivatives	Off	{On,Off}
8)	USEG	Use/Recalculate gradient	0	{1,0}
9)	USEH	Use/Recalculate Hessian	0	{1,0}
10)	LS	Line search conditions	Weak	{Strong,Weak}
11)	LSITER	Max. LS iterations	30	Any int >= 1
12)	RHO	Line search rho	0.0001	0. <= real <= 1.
13)	SIGMA	Line search sigma	0.9	0. <= real <= 1.
14)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.6: The `DFP` panel.

- 2 The gradient criterion is satisfied.
- 3 All function evaluations have been used.
- 4 The X-convergence criterion is satisfied.
- 5 The function value has converged.
- 6 The Hessian matrix cannot be updated.
- 7 The specified number of iterations has been reached.
- 8 All variables are fixed.
- 9 Further progress is not possible.

6.3.4 `DFP`

Syntax classification: PAC

Purpose: Invokes the DFP minimization algorithm using line search and Choleski decomposition for the Hessian. The associated panel is listed in 6.6. The panel parameters are the same as for the `BFGS` command.

6.3.5 `TOLMIN`

Syntax classification: PAC

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	ACC	Termination accuracy	0.	0. <= real < 1.
4)	MAD	Automatic derivatives	Off	{On,Off}
5)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.7: The `TOLMIN` panel.

Purpose: Invokes the BFGS minimization algorithm using line search and the Goldfarb–Idnani $\boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z} = I$ decomposition for the Hessian. The associated panel is listed in figure 6.7. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`PRINT` Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.
- 2 Display function and parameter values as well.

`ACC` Specifies the termination accuracy.

`MAD` Specifies whether the Merlin Automatic Derivatives are to be used when approximating the gradient vector. Allowed values are:

- `ON` The Merlin Automatic Derivatives are to be used.
- `OFF` Use the current derivative mode.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to Mcl programs:

`FCALLS` Number of function evaluations that were performed.

`GCALLS` Number of evaluations of the gradient vector.

`INFO` Result code. Possible values are:

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	GTOL	G-convergence criterion	1.E-15	0. < real < 1.
4)	XTOL	X-convergence criterion	1.E-15	0. < real < 1.
5)	FTOL	F-convergence criterion	0.	0. <= real < 1.
6)	ITER	Iterations (-1 = Inf)	-1	Any int >= -1
7)	MAD	Automatic derivatives	Off	{On,Off}
8)	USEG	Use/Recalculate gradient	1	{1,0}
9)	USEH	Use/Recalculate Hessian	0	{1,0}
10)	USER	Use/Recalculate radius	0	{1,0}
11)	RHO	The rho-line parameter	0.0001	0. <= real <= 1.
12)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.8: The `TRUST` panel.

- 1 The specified accuracy has been satisfied.
- 2 Further progress is impossible due to rounding errors.
- 3 Further progress is impossible.
- 4 Some input parameters are incorrect.
- 5 Inconsistent equality constraints.
- 6 Inconsistent constrains/bounds.
- 7 All of the constrains cannot be satisfied.
- 8 All function evaluations have been used.
- 9 Target value has been reached.

6.3.6 `TRUST`

Syntax classification: PAC

Purpose: Invokes the BFGS minimization algorithm using a trust region method and Choleski decomposition for the Hessian. The associated panel is listed in figure 6.8. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`PRINT` Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.

- 2 Display function and parameter values as well.

`GTOL` The gradient termination criterion. The method terminates when the relative gradient falls below `GTOL`.

`XTOL` Termination criterion based on the values of the parameters. The method terminates when the relative change in the parameters in two successive iterations is less than `XTOL`.

`FTOL` Termination criterion based on function values. The method terminates when the relative function drop in two successive iterations is less than `FTOL`.

`ITER` The number of BFGS iterations to perform. A value of -1 allows an unlimited number of iterations.

`MAD` Specifies whether the MERLIN Automatic Derivatives are to be used when approximating the gradient vector. Allowed values are:

- `ON` The MERLIN Automatic Derivatives are to be used.
- `OFF` Use the current derivative mode.

`USEG` Determines how to obtain the gradient vector for the initial (current) point. Allowed values are:

- 0 Do not use the values in the gradient cache. Recalculate the whole gradient vector.
- 1 Instead of recalculating the gradient vector, use the values in the gradient cache.

`USEH` Determines the initial approximation to the Hessian matrix. Allowed values are:

- 0 The initial approximation is set to be the unit matrix.
- 1 The approximation from a previous `BFGS`, `DFP`, `TRUST` or `HESSIAN` command is used.

`USER` Determines the initial trust region radius. Allowed values are:

- 0 The trust region radius is set equal to $\frac{1}{10}||\boldsymbol{g}||$
- 1 The most recent value for the radius is used.

`RHO` The ρ parameter used in the termination criteria for the line search.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

FCALLS Number of function evaluations that were performed.

GCALLS Number of evaluations of the gradient vector.

ITERDONE Number of BFGS iterations that were performed.

RADIUS The final trust region radius.

INFO Result code. Possible values are:

- 1 The target value has been reached.
- 2 The gradient criterion is satisfied.
- 3 All function evaluations have been used.
- 4 The minimization parameters have converged.
- 5 The function value has converged.
- 6 The Hessian matrix cannot be updated.
- 7 The specified number of iterations has been reached.
- 8 All variables are fixed.
- 9 Further progress is not possible.

6.3.7 CONGRA

Syntax classification: PAC

Purpose: Invokes any of the three implemented conjugate gradient algorithms. These are:

- The Polak–Ribiere algorithm.
- The Fletcher–Reeves algorithm.
- The Generalized Polak–Ribiere algorithm.

The associated panel is listed in figure 6.9. The panel parameters are as follows:

NOC An approximate upper bound for the number of calls to the objective function.

PRINT Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.
- 2 Display function and parameter values as well.

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	METHOD	Conjugate gradient method	PR	{PR,FR,GPR}
4)	GTOL	G-convergence criterion	1.E-15	0. < real < 1.
5)	XTOL	X-convergence criterion	1.E-15	0. < real < 1.
6)	FTOL	F-convergence criterion	0.	0. <= real < 1.
7)	ITER	Iterations (-1 = Inf)	-1	Any int >= -1
8)	RESTART	N-step restart (No/Yes)	0	{0,1}
9)	MAD	Automatic derivatives	Off	{On,Off}
10)	USEG	Use/Recalculate gradient	1	{1,0}
11)	LSITER	Max. LS iterations	30	Any int >= 1
12)	RHO	Line search rho	0.001	0. <= real <= 1.
13)	SIGMA	Line search sigma	0.1	0. <= real <= 1.
14)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.9: The `CONGRA` panel.

`METHOD` Determines the method to use. Allowed values are:

- `PR` Use the Polak–Ribiere formula.
- `FR` Use the Fletcher–Reeves formula.
- `GPR` Use the generalized Polak–Ribiere method of Khoda, Liu and Storey.

`GTOL` The gradient termination criterion. The method terminates when the relative gradient falls below `GTOL`.

`XTOL` Termination criterion based on the values of the parameters. The method terminates when the relative change in the parameters in two successive iterations is less than `XTOL`.

`FTOL` Termination criterion based on function values. The method terminates when the relative function drop in two successive iterations is less than `FTOL`.

`ITER` The number of iterations to perform. A value of -1 allows an unlimited number of iterations.

`RESTART` Specifies whether the N–step restart should be taken. Allowed values are:

- 0 Do not use the N–step restart.
- 1 Reset the search direction to the steepest descent direction every N iterations.

`MAD` Specifies whether the Merlin Automatic Derivatives are to be used when approximating the gradient vector. Allowed values are:

- `ON` The Merlin Automatic Derivatives are to be used.

- `OFF` Use the current derivative mode.

`USEG` Determines how to obtain the gradient vector for the initial (current) point. Allowed values are:

- 0 Do not use the values in the gradient cache. Recalculate the whole gradient vector.
- 1 Instead of recalculating the gradient vector, use the values in the gradient cache.

`LSITER` The maximum number of iterations that are allowed in the line search.

`RHO` The ρ parameter used in the termination criteria for the line search.

`SIGMA` The σ parameter used in the termination criteria for the line search.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

`FCALLS` Number of function evaluations that were performed.

`GCALLS` Number of evaluations of the gradient vector.

`ITERDONE` Number of conjugate gradient iterations that were performed.

`INFO` Result code. Possible values are:

- 1 Target value has been reached.
- 2 The gradient criterion is satisfied.
- 3 All function evaluations have been used.
- 4 The X-convergence criterion is satisfied.
- 5 The function value has converged.
- 7 The specified number of iterations has been reached.
- 8 Further progress is not possible.
- 9 All variables are fixed.

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	PRINT	Printout level	1	{0,1,2}
3)	GTOL	G-tolerance	0.	0. <= real < 1.
4)	XTOL	X-tolerance	0.	0. <= real < 1.
5)	FTOL	F-tolerance	0.	0. <= real < 1.
6)	FACC	Function accuracy	1.E-15	0. < real < 1.
7)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.10: The LEVE panel.

6.3.8 LEVE

Syntax classification: PAC

Purpose: Invokes a Levenberg–Marquardt type of algorithm for a sum of squares objective function. The associated panel is listed in figure 6.10. The panel parameters are as follows:

NOC An approximate upper bound for the number of calls to the objective function.

PRINT Determines the amount of output from the minimization method. Allowed values are:

- 0 No printout at all.
- 1 Display lower function values as they are discovered.
- 2 Display function and parameter values as well.

GTOL The gradient termination criterion. The method terminates when the relative gradient falls below GTOL.

XTOL Termination criterion based on the values of the parameters. The method terminates when the relative change in the parameters in two successive iterations is less than XTOL.

FTOL Termination criterion based on function values. The method terminates when the relative function drop in two successive iterations is less than FTOL.

FACC An estimation of the relative error in the calculation of the objective function.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	10000	Any int >= 1
2)	TARGET	Target value	-1.E+300	Any real
3)	MAD	Automatic derivatives	Off	{On,Off}
4)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.11: The `AUTO` panel.

`FCALLS` Number of function evaluations that were performed.

`JCALLS` Number of evaluations of the Jacobian matrix.

`ITERDONE` Number of `LEVE` iterations that were performed.

`INFO` Result code. Possible values are:

- 0 Some input parameters were incorrect.
- 1 The function value has converged.
- 2 The X-convergence criterion is satisfied.
- 3 F and X-convergence has been achieved.
- 4 The gradient criterion is satisfied.
- 5 All function evaluations have been used.
- 6 Further progress is not possible.
- 7 Further progress is not possible.
- 8 Further progress is not possible.
- 9 Target value has been reached.

6.3.9 `AUTO`

Syntax classification: PAC

Purpose: Invokes a multi algorithm based strategy. The associated panel is listed in figure 6.11. The panel parameters are as follows:

`NOC` An approximate upper bound for the number of calls to the objective function.

`TARGET` Specifies a target value. When the value of the objective function falls below this value, the `AUTO` procedure terminates.

`MAD` Specifies whether the Merlin Automatic Derivatives are to be used when approximating the gradient vector. Allowed values are:

Ind	Keyword	Description	Value	Allowed values
1)	NOC	Number of calls	300	Any int >= 1
2)	NOP	Number of points	1	Any int >= 1
3)	TARGET	Target value	16.913	Any real
4)	FILE	File name	INIPO	Any string
5)	TYPE	File type	Text	{Text,Bin}
6)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.12: The `ACCUM` panel.

- `ON` The MERLIN Automatic Derivatives are to be used.
- `OFF` Use the current derivative mode.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

`FCALLS` Number of function evaluations that were performed.

`GCALLS` Number of evaluations of the gradient vector.

`INFO` Result code. Possible values are:

- 1 All function evaluations have been used.
- 2 Further progress is not possible.
- 3 Target value has been reached.

6.3.10 `ACCUM`

Syntax classification: PAC

Purpose: Picks points at random and stores them in a file if the corresponding function value is lower than a preset number. The associated panel is listed in figure 6.12.

Restriction: All free variables must be bounded from both above and below. The panel parameters are as follows:

`NOC` An upper bound to the number of calls to the objective function.

NOP The number of points to accumulate.

TARGET Sets the target value. The points $\boldsymbol{x}$ for which $f(\boldsymbol{x}) \leq$ `TARGET` are stored in the file specified by the `FILE` keyword.

FILE The file to store the accumulated points.

TYPE The file type. Allowed values are:

- `TEXT` for a MERLIN text file. Information is stored in human–readable, ASCII form. Slower but portable.
- `BIN` for a MERLIN binary file. Information is stored in binary, machine–dependent form. Faster, but less portable.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

FCALLS Number of function evaluations that were performed.

POINTS Number of points that have been accumulated.

INFO Result code. Possible values are:

- 0 The command terminated normally.
- −1 An end–of–file condition occurred in the input file, while reading the panel parameters.
- −2 Some other error occurred (for example some panel parameters are incorrect).

6.3.11 TARGET

Syntax classification: UNC

Syntax: `TARGET` *target_value*

Purpose: Sets the target value used as a termination criterion by the optimization algorithms.

6.3.12 `NOTARGET`

Syntax classification: SIC

Purpose: Clears a target value earlier set by a `TARGET` command.

6.3.13 `ADJUST`

Syntax classification: SIC

Purpose: Constructs search steps for the `ROLL` method without using any gradient information. It is based on the alternating variables method. In addition `ADJUST` performs a minor optimization.

6.3.14 `STEPALL`

Syntax classification: SIC

Purpose: Constructs search steps for the `ROLL` method using gradient information. It is based on the method of steepest descent. In addition `STEPALL` performs a minor optimization.

6.3.15 `STEP`

Syntax classification: RAVAC

Purpose: Facilitates setting the search steps for the `ROLL` method.

Examples: To set the step for the x_1 variable to 0.1 and for the variables x_3 through x_6 to 2.5, use the following:

```
STEP 1 0.1 3-6 2.5
```

6.3.16 `STEPDIS`

Syntax classification: RAC

Purpose: Displays the current search steps for the `ROLL` method.

6.4 Flags and related commands

6.4.1 `FLAG`

Syntax classification: RAVAC

Purpose: Assigns values to the numeric MERLIN flags. The default value for these flags is zero.

6.4.2 `FLAGDIS`

Syntax classification: RAC

Purpose: Displays the values of the specified numerical MERLIN flags. Without arguments the values of all MERLIN numerical flags are displayed.

6.4.3 `CFLAG`

Syntax classification: RAVAC

Purpose: Assigns values to the MERLIN character flags. The default value for these flags is a blank character.

6.4.4 `CFLAGDIS`

Syntax classification: RAC

Purpose: Displays the values of the specified MERLIN character flags. Without arguments the values of all MERLIN character flags are displayed.

6.5 Modes of operation and related commands

6.5.1 `FAST`, `QUAD`, `NUMER`, `ANAL`

Syntax classification: SIC

Purpose: Set the way the gradient vector is calculated

`FAST` Forward differences are used to estimate the derivatives. This costs one function evaluation per gradient component:

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x)}{h} \quad \text{with} \quad h = \sqrt{\eta}\max\{1, |x|\}$$

`QUAD` Central differences are used to estimate the derivatives. This costs two function evaluations per gradient component:

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x-h)}{2h} \quad \text{with} \quad h = \sqrt[3]{\eta}\max\{1, |x|\}$$

NUMER A high order symmetric formula is used to estimate the derivatives. This costs at least six function evaluations per gradient component:

$$\frac{df}{dx} \approx \frac{64}{45}\frac{f(x+h)-f(x-h)}{2h} - \frac{20}{45}\frac{f(x+2h)-f(x-2h)}{4h} + \frac{1}{45}\frac{f(x+4h)-f(x-4h)}{8h}$$

ANAL The user written code (SUBROUTINE GRANAL) is used to calculate the derivatives.

6.5.2 MIXED

Syntax classification: RAVAC

Purpose: Allows different modes to be assigned to specified gradient components.

Examples: To calculate the first 3 components in ANAL mode, the fourth in NUMER, etc. use:

```
MIXED 1-3 ANAL 4 NUMER 5 FAST 6 QUAD
```

or abbreviating:

```
MIXED 1-3 A 4 N 5 F 6 Q
```

in an obvious correspondence.

6.5.3 JNUMER, JANAL

Syntax classification: SIC

Purpose: Make a choice for the manner the Jacobian is calculated.

JNUMER selects numerical estimation using forward differences.

JANAL selects the user supplied code (SUBROUTINE JANAL).

6.5.4 HESSIAN

Syntax classification: PAC

Purpose: Calculates and manipulates the Hessian matrix. The associated panel is listed in figure 6.13. The panel parameters are as follows:

DO Determines the action of the command. Allowed values are:

- C Calculate the Hessian matrix.
- D Decompose an already calculated Hessian matrix to its Choleski factors.
- M Compose the Hessian matrix from its Choleski factors.

Ind	Keyword	Description	Value	Allowed values
1)	DO	Desired Action	c	{c,d,m,p,r,w}
2)	USE	Function/Gradient/Anal	f	{f,g,a}
3)	MODE1	Function/Gradient/Anal	f	{f,g,a}
4)	MODE2	Function/Gradient/Anal	g	{f,g,a}
5)	FILE	File to write to	HESSIAN	Any string
6)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.13: The `HESSIAN` panel.

- `P` Compare two methods of calculating the Hessian matrix.
- `R` Read the Hessian matrix from a file.
- `W` Write the Hessian matrix to a file.

`USE` If the calculation of the Hessian was requested, `USE` determines the method of calculation. Allowed values are:

- `F` Use function values only:

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} \approx \frac{[f(x+h,y+\delta) - f(x+h,y)] - [f(x,y+\delta) - f(x,y)]}{h\delta}$$

  For the diagonal elements:

$$\frac{d^2 f}{dx^2} \approx \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}$$

  with $h = \sqrt[3]{\eta}\max\{1,|x|\}$ and similarly for δ.

- `G` Use the user supplied `SUBROUTINE GRANAL`:

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} \approx \frac{1}{2}\left\{\left[\frac{\partial f(x,y+\delta)}{\partial x} - \frac{\partial f(x,y)}{\partial x}\right]\frac{1}{\delta} + \left[\frac{\partial f(x+h,y)}{\partial y} - \frac{\partial f(x,y)}{\partial y}\right]\frac{1}{h}\right\}$$

  For the diagonal elements:

$$\frac{d^2 f}{dx^2} \approx \frac{1}{h}\left[\frac{\partial f(x+h)}{\partial x} - \frac{\partial f(x)}{\partial x}\right]$$

  with $h = \sqrt{\eta}\max\{1,|x|\}$ and similarly for δ.

- `A` Use the user supplied `SUBROUTINE HANAL`.

`MODE1` If comparison of two methods for calculating the Hessian matrix was requested, `MODE1` specifies the first mode of calculation. Allowed values are:

- `F` Use function values only.
- `G` Use the the user supplied `SUBROUTINE GRANAL` that returns the first partial derivatives.

- `A` Use the user supplied `SUBROUTINE HANAL` that returns the Hessian matrix.

`MODE2` If comparison of two methods for calculating the Hessian matrix was requested, `MODE2` specifies the second mode of calculation. Allowed values are:

- `F` Use function values only.
- `G` Use the the user supplied `SUBROUTINE GRANAL` that returns the first partial derivatives.
- `A` Use the user supplied `SUBROUTINE HANAL` that returns the Hessian matrix.

`FILE` The file to write to, or read from the Hessian matrix.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.5.5 GRADDIS

Syntax classification: RAC

Purpose: Displays the values of the gradient components and the mode used for their calculation.

Examples: `GRADDIS !/F`

6.5.6 GRADCHECK

Syntax classification: UNC

Syntax: `GRADCHECK` *mode*$_1$ *mode*$_2$ [*index_specification* ...]
`GRADCHECK` *mode* [*index_specification* ...]

Purpose: Checks the derivatives calculated using *mode*$_1$ against those calculated using *mode*$_2$ and displays the result for the parameters specified by the *index_specification*. If the second form of the command is used, the derivatives are checked against the current derivative mode. *mode*$_1$, *mode*$_2$ and *mode* can be one of `ANAL`, `FAST`, `QUAD` or `NUMER`. If no *index_specification* is given, the results are displayed for all minimization parameters.

Examples:
```
GRADCHECK ANAL QUAD
GRADCHECK ANAL NUMER 1-6
```

Ind	Keyword	Description	Value	Allowed values
1)	FQ	Fast ==> Quad threshold	0.0001	0. <= real < 1.
2)	QN	Quad ==> Numer threshold	1.E-7	0. <= real < 1.
3)	RECALC	Recalculate on change	1	{0,1}
4)	PRINT	MAD printout (Off/On)	0	{0,1}

Figure 6.14: The `MAD` panel.

6.5.7 `JCOMPARE`

Syntax classification: SIC

Purpose: Compares the numerically estimated Jacobian to the one calculated by the user–supplied code.

6.5.8 `GNORM`

Syntax classification: SIC

Purpose: Calculates the L_1, L_2 and L_∞ gradient norms, along with the RMS gradient, defined as:

$$L_1 = \sum_{i=1}^{N} |g_i|$$

$$L_2 = \sqrt{\sum_{i=1}^{N} g_i^2}$$

$$L_\infty = \max |g_i|, \quad i = 1, 2, \ldots N$$

$$g_{\mathrm{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} g_i^2}$$

6.5.9 `MAD`

Syntax classification: PAC

Purpose: Sets the thresholds for automatically switching derivative modes from `FAST` to `QUAD` and from `QUAD` to `NUMER`. The associated panel is listed in figure 6.14. The panel parameters are as follows:

`FQ` The `FAST` to `QUAD` transition threshold. When a gradient component that is calculated using mode `FAST`, falls below this value, its mode is set to `QUAD`.

`QN` The `QUAD` to `NUMER` transition threshold. When a gradient component that is calculated using mode `QUAD`, falls below this value, its mode is set to `NUMER`.

`RECALC` Determines whether a recalculation of the partial derivative should occur, each time a mode transition takes place. Allowed values are:

- 0 Do not recalculate, each time a transition occurs.
- 1 Recalculate using the new derivative mode.

`PRINT` Determines whether the user should be notified when a change in the gradient mode occurs. Allowed values are:

- 0 No informative messages.
- 1 Display an informative message each time a gradient mode changes. This can be annoying, especially for an objective function with a large number of parameters.

6.5.10 `IAF`, `BATCH`

Syntax classification: SIC

Purpose: `IAF` sets an error tolerant mode, suitable for interactive work. `BATCH` sets the strict mode, that causes MERLIN to abort on errors and is suitable for unattended (batch) processing.

6.5.11 `NOBACK`, `LASTBACK`, `FULLBACK`

Syntax classification: SIC

Purpose: Set the backup mode.

`NOBACK` keeps no backup records.

`LASTBACK` keeps only the most recent record.

`FULLBACK` keeps a history of all records resulting after the completion of the issued minimization commands.

A backup record contains the values of the attributes of the minimization parameters.

Ind	Keyword	Description	Value	Allowed values
1)	FILE	Backup file name	BACKUP	Any string
2)	TYPE	File type	Text	{Text,Bin}
3)	MODE	Backup mode	No	{Full,Last,No}
4)	WHEN	When to Backup (MXDL)	mxd	Any string
5)	SAFE	Safe (but slow) backup	No	{Yes,No}
6)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.15: The `BACKUP` panel.

6.5.12 `BACKUP`

Syntax classification: PAC

Purpose: Controls the MERLIN backup mechanism. Prevents loss of data from abnormal program termination. The associated panel is listed in figure 6.15. The panel parameters are as follows:

`FILE` The backup file name.

`TYPE` The file type. Allowed values are:

- `TEXT` for a MERLIN text file. Information is stored in human–readable, ASCII form. Slower but portable.
- `BIN` for a MERLIN binary file. Information is stored in binary, machine–dependent form. Faster, but less portable.

`MODE` Specifies the amount of information that is kept in a backup file. Allowed values are:

- `NO` Backups are not performed.
- `LAST` The most recent backup record overwrites the previous contents of the backup file.
- `FULL` The backup records are appended to the end of the backup file.

`WHEN` Specifies when a backup record is to be written to the backup file. Allowed values are:

- `M` A backup record is written when a minimization routine terminates.
- `X` A backup record is written each time the minimization parameters are assigned new values (through `POINT`, `INIT` or `PICK`).
- `D` A backup record is written each time the fix status, symbolic name, upper or lower bounds of a parameter is changed.

- `L` A backup record is written each time a minimization routine discovers a new lower function value. This setting is only recommended for extremely time consuming minimization sessions.

Any combination of the above values is allowed.

`SAFE` Specifies whether the slow (but safe) backup method should be used. Allowed values are:

- `YES` Each time a backup record is written to the backup file, its contents are flushed to disk, thus preventing loss of backup records if the program terminates abnormally.
- `NO` Just write the backup records to the file.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.5.13 `NOPRINT`, `HALFPRINT`, `FULLPRINT`

Syntax classification: SIC

Purpose: Set the printout mode.

`FULLPRINT` allows the full output to be displayed.

`HALFPRINT` allows only warnings and error messages.

`NOPRINT` suppresses all output.

6.5.14 `GENERAL`, `SOS`

Syntax classification: SIC

Purpose: Determine which form for the objective function is to be used.

`GENERAL` Makes calls the `FUNCTION FUNMIN`.

`SOS` Makes calls to `SUBROUTINE SUBSUM`.

The default setting depends on the number of terms M, supplied when `SUBROUTINE MERLIN` is called. When $M = 0$ the default is set to `GENERAL`; `SOS` otherwise.

6.5.15 EVALUATE, NOEVAL

Syntax classification: SIC

Purpose: NOEVAL disallows evaluation of the objective function. EVALUATE allows the evaluation of the objective function, if it was disallowed, and in addition forces one evaluation.

6.5.16 MODEDIS

Syntax classification: SIC

Purpose: Displays the current operation modes.

6.5.17 LIMITS

Syntax classification: SIC

Purpose: Displays the values of several installation parameters.

6.6 Aliasing and related commands

6.6.1 ALIAS

Syntax classification: UNC

Syntax:ALIAS *alias_name command*

Purpose: Renames a command for the current run.

Examples: To create an alias for the command GRADDIS use:
ALIAS GD GRADDIS.

6.6.2 UNALIAS

Syntax classification: UNC

Syntax: UNALIAS *alias_name*

Purpose: Clears an alias entry.

Examples: To clear the previously defined alias GD for the command GRADDIS use:
UNALIAS GD

6.6.3 `ALIASDIS`

Syntax classification: SIC

Purpose: Prints out all the aliases and their command equivalences.

6.7 Termination and post–processing

6.7.1 `STOP`

Syntax classification: UNC

Syntax: `STOP` [`NOEPILOG`]

Purpose: Terminates the current run, by issuing a Fortran `STOP` statement.

6.7.2 `RETURN`

Syntax classification: UNC

Syntax: `RETURN` [`NOEPILOG`]

Purpose: Returns the control to the calling program (i.e. to the program that calls `SUBROUTINE MERLIN`). It executes a Fortran `RETURN` statement.

6.7.3 `QUIT`

Syntax classification: UNC

Syntax: `QUIT` *integer_flag* [`NOEPILOG`]

Purpose: It sets a value to the output flag `IQUIT` of `SUBROUTINE MERLIN` and then executes a Fortran `RETURN` statement. The allowed values for the *return_flag* are positive. The `RETURN` command assigns a zero value to this flag without prompting. This flag can be used by the main program as a directive for a required user–programmed action, referred to as post-processing. All the above termination commands may take an optional argument that inhibits the execution of an epilog, if one exists.

Examples:
```
QUIT 4
STOP NOEPILOG
RETURN NOEPILOG
QUIT 7 NOEPILOG
```

Ind	Keyword	Description	Value	Allowed values
1)	FILE	Memo file name	STORE	Any string
2)	TYPE	File type	Text	{Text,Bin}
3)	OPEN	Keep memo file open	0	{0,1}
4)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.16: The `MEMO` panel.

6.8 File manipulation commands

6.8.1 `MEMO`

Syntax classification: PAC

Purpose: Appends a MERLIN record with the values and attributes of the minimization parameters to a specified file. The associated panel is listed in figure 6.16. The panel parameters are as follows:

`FILE` The file to write to.

`TYPE` The file type. Allowed values are:

- `TEXT` for a MERLIN text file. Information is stored in human–readable, ASCII form. Slower but portable.
- `BIN` for a MERLIN binary file. Information is stored in binary, machine–dependent form. Faster, but less portable.

`OPEN` Specifies whether the `MEMO` file should remain open after the `MEMO` command completes. Allowed values are:

- 0 Close the memo file.
- 1 Leave the memo file open. Subsequent `MEMO` commands will run faster but will be less safe against a system failure.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

Ind	Keyword	Description	Value	Allowed values
1)	FILE	File to write to	DUMP	Any string
2)	WHAT	What to write	x	{x,l,r,f,s,n}
3)	NPL	Numbers per line	0	Any int >= 0
4)	APPEND	Append to the file	No	{Yes,No}
5)	FROM	Index to start from	1	Any int >= 1
6)	TO	Index to end to	2	Any int >= 1
7)	VALUE	Write value	Never	{Before,After,Never}
8)	FORMAT	Format, eg: E20.10	*	Any string
9)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.17: The `DUMP` panel.

6.8.2 `DUMP`

Syntax classification: PAC

Purpose: Stores the current values of the minimization parameters or any of their attributes in a file. The associated panel is listed in figure 6.17. The panel parameters are as follows:

`FILE` The file to write to.

`WHAT` Specifies what to write. Allowed values are:

- `X` Write the current point.
- `L` Write the left margins.
- `R` Write the right margins.
- `F` Write the fix statuses (1 for a free variable, 0 for a fixed one).
- `S` Write the search steps used by the `ROLL` command.
- `N` Write the symbolic names of the parameters.

`NPL` Specifies how many numbers will be written on each line of the file. This does not apply to symbolic names, which are always written one per line. Setting `NPL` to 0, lets the system decide for the actual numbers per line.

`APPEND` In case the specified file exists, `APPEND` determines what to do with its previous contents. Allowed values are:

- `NO` The previous contents of the file are overwritten.
- `YES` The new values are appended at the end of the file.

`FROM` Specifies the index to start writing from.

TO Specifies the index to end writing to.

VALUE Specifies whether the value of the objective function is to be written. Allowed values are:

- BEFORE The value of the objective function is written before the actual data.
- AFTER The value of the objective function is written after the actual data.
- NEVER The value of the objective function is not written.

FORMAT This is the format to be used when writing data to the file. You may specify any valid Fortran format, or * to use a free format. Note that the format must be consistent with the setting of NPL, containing the appropriate number of format specifiers. Symbolic names are always written one per line using an A format.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.8.3 DISCARD

Syntax classification: UNC

Syntax: DISCARD *file_name* [TEXT | BIN]

Purpose: Writes a MERLIN record with the values and attributes of the minimization parameters to a specified file. The file is overwritten. The optional parameter at the end of the command specifies the file type.

Examples: `DISCARD DATAFILE`
overwrites the file DATAFILE with the values and attributes of the minimization parameters.

6.8.4 DELETE

Syntax classification: UNC

Syntax: DELETE *file_name*$_1$ *file_name*$_2$ ...

Purpose: Deletes one or more files.

Examples: `DELETE DATAFILE store`
deletes the files DATAFILE and store.

Ind	Keyword	Description	Value	Allowed values
1)	FILE	File name to inspect	BACKUP	Any string
2)	TYPE	File type	Text	{Text,Bin}
3)	DIR	Direction (Forw./Backw.)	1	{1,-1}
4)	PRINT	Print Nothing/Value/Point	1	{0,1,2}
5)	FROM	Record to start from	0	Any int >= 0
6)	TO	Record to end to	0	Any int >= 0
7)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.18: The `INSPECT` panel.

6.8.5 `REWIND`

Syntax classification: UNC

Syntax: `REWIND` *file_name*$_1$ *file_name*$_2$ ...

Purpose: Rewinds one or more files. Mainly used in MCL programs.

Examples: `REWIND DATAFILE store`
rewinds the files `DATAFILE` and `store`.

6.8.6 `GOEOF`

Syntax classification: UNC

Syntax: `GOEOF` *file_name*$_1$ *file_name*$_2$ ...

Purpose: Positions one or more files to the end of information. Mainly used in MCL programs.

Examples: `GOEOF DATAFILE store`
positions the files `DATAFILE` and `store` to the end of information.

6.8.7 `INSPECT`

Syntax classification: PAC

Purpose: Displays the contents of a file containing MERLIN records (stored by the backup mechanism or the `MEMO` command). Inspection can proceed forwards or backwards and can display parameter values, or function values only. The associated panel is listed in figure 6.18. The panel parameters are as follows:

`FILE` The file to inspect.

TYPE The file type. Allowed values are:

- **TEXT** for a MERLIN text file. Information is stored in human–readable, ASCII form. Slower but portable.
- **BIN** for a MERLIN binary file. Information is stored in binary, machine–dependent form. Faster, but less portable.

DIR Determines the direction of the inspection. Allowed values are:

- 1 The file is inspected forwards, from the beginning to the end.
- −1 The file is inspected backwards, from the end to the beginning.

The settings of FROM, TO also affect the direction of inspection. For a forward inspection FROM should be less or equal to TO, while for a backward inspection FROM should be greater than or equal to TO. If these conditions do not hold, the selected direction is automatically inverted.

PRINT Determines the amount of output during file inspection. Allowed values are:

- 0 No printout at all.
- 1 Display function values.
- 2 Display function and parameter values as well.

FROM The record where inspection starts.

TO The record where inspection ends.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

The following parameters can be returned to MCL programs:

REC Index of the last record that has been inspected.

6.9 Graphics and related commands

6.9.1 GRAPH

Syntax classification: PAC

Ind	Keyword	Description	Value	Allowed values
1)	INDEX	Variable index	1	Any int >= 1
2)	NOP	Number of points	30	Any int >= 1
3)	FROM	From	-10000.	Any real
4)	TO	To	10000.	Any real
5)	LINES	Lines	17	Any int >= 1
6)	COLS	Columns	64	Any int >= 1
7)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.19: The `GRAPH` panel.

Purpose: Displays a one–dimensional rough graph of the objective function with respect to one of its parameters. The associated panel is listed in figure 6.19. The panel parameters are as follows:

`INDEX` The function is graphed with respect to this variable.

`NOP` Number of points for the graph. Each point corresponds to an evaluation of the objective function.

`FROM` Along with `TO`, this parameter determines the interval over which the function is displayed. This interval refers to the variable specified by `INDEX`.

`TO` Along with `FROM`, this parameter determines the interval over which the function is displayed. This interval refers to the variable specified by `INDEX`.

`LINES` The number of lines to use.

`COLS` The number of columns to use.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.9.2 `PSGRAPH`

Syntax classification: PAC

Purpose: Produces a Postscript graph of the objective function, or of a set of X–Y data points. The associated panel is listed in figure 6.20. The panel parameters are as follows:

`WHAT` Specifies what to plot. Allowed values are:

Ind	Keyword	Description	Value	Allowed values
1)	WHAT	What to plot (Func/Data)	f	{f,d}
2)	FILE	Output file name	POST	Any string
3)	INDEX	Variable index	1	Any int >= 1
4)	NOP	Number of points	30	1 <= int <= 1000
5)	FROM	From	-1000.	Any real
6)	TO	To	1000.	Any real
7)	DATAFILE	Data file to read	DATA	Any string
8)	SYMBOL	Symbol (No/Yes)	1	{0,1}
9)	XTITLE	X-axis title		Any string
10)	YTITLE	Y-axis title		Any string
11)	FONTSIZE	X and Y-title fontsize	16	5 <= int <= 30
12)	GRID	Grid lines (On/Off)	0	{1,0}
13)	FRAME	Frame (On/Off)	1	{1,0}
14)	CONNECT	Connect points (Yes/No)	1	{1,0}
15)	XTICKS	Number of X-axis ticks	5	0 <= int <= 15
16)	YTICKS	Number of Y-axis ticks	5	0 <= int <= 15
17)	PAPER	Paper (A4/B5/USLet/USLeg)	0	{0,1,2,3}
18)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.20: The `PSGRAPH` panel.

- `F` Plots the objective function with respect to one of its variables.
- `D` Plots X–Y curves from data points stored in a file.

`FILE` The file to dispose the PostScript graph.

`INDEX` The objective function is plotted with respect to this variable.

`NOP` Number of points (evaluations) in case one wants to plot the objective function.

`FROM` Along with `TO`, this parameter determines the interval over which the function is plotted. This interval refers to the variable specified by `INDEX`.

`TO` Along with `FROM`, this parameter determines the interval over which the function is plotted. This interval refers to the variable specified by `INDEX`.

`DATAFILE` In case one wants to plot a set of data points, this is the file containing these points. The points should be stored column wise, with the first column being the X–values and the rest of the columns being sets of Y–values that correspond to different curves. Spaces or tabs can be used to separate the numbers in a line.

`SYMBOL` Specifies whether a symbol should be plotted at each data point. Allowed values are:

- 0 Do not plot a symbol.

- 1 Plot a symbol (an $\times$) at each data point.

`XTITLE` The X–axis title.

`YTITLE` The Y–axis title.

`FONTSIZE` The X and Y–axis titles are printed using the Times–Roman font and this font size.

`GRID` Specifies whether grid lines should be plotted. Allowed values are:

- 0 Do not plot grid lines.
- 1 Plot grid lines.

`FRAME` Specifies whether a frame should be plotted around the graph. Allowed values are:

- 0 Do not plot a frame around the graph.
- 1 Plot a frame around the graph.

`CONNECT` Specifies whether the data points should be connected by a straight line or not. Allowed values are:

- 0 Do not connect the data points.
- 1 Connect the data points using straight lines.

`XTICKS` Number of X–axis ticks.

`YTICKS` Number of Y–axis ticks.

`PAPER` Specifies the paper size. Allowed values are:

- 0 A4 paper size (8.26" $\times$ 11.69").
- 1 B5 paper size (7.16" $\times$ 10.12).
- 2 US Letter paper size (8.5" $\times$ 11").
- 3 US Legal paper size (8.5" $\times$ 14").

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.10 Panel related commands

6.10.1 `PANELON`

Syntax classification: UNC

Syntax: **`PANELON`** [*panel_name*$_1$ *panel_name*$_2$ ...]

Purpose: Activates one or more panels. Without any arguments it turns on all MERLIN panels.

Examples: **`PANELON ROLL BFGS PICK`**

6.10.2 `PANELOFF`

Syntax classification: UNC

Syntax: **`PANELOFF`** [*panel_name*$_1$ *panel_name*$_2$ ...]

Purpose: Deactivates one or more panels. Without any arguments it turns off all MERLIN panels.

Examples: **`PANELOFF BFGS PICK`**

6.10.3 `PSTATUS`

Syntax classification: UNC

Syntax: **`PSTATUS`** [*panel_name*$_1$ *panel_name*$_2$ ...]

Purpose: Displays the status of some or all MERLIN panels. A panel can be active (on) or inactive (off). Without any arguments the status of all MERLIN panels is displayed.

6.10.4 `PDUMP`

Syntax classification: UNC

Syntax: **`PDUMP`** *file_name* [*panel_name*$_1$ *panel_name*$_2$ ...]

Purpose: Writes to a file the current values of the parameters of one or more panel commands. The file can be used later as a macro to restore the values of the panel parameters. If no panel names are present in the command line, the parameters of all MERLIN panel commands are written to the file.

6.11 Output redirection and related commands

Almost all MERLIN commands produce output. This output can be redirected to a file using the Unix like `>` and `>>` redirection symbols:

command argument$_1$ *argument*$_2$ ... `>` *file_name*

will dispose the command's output to the file *file_name*. The file is overwritten.

command argument$_1$ *argument*$_2$ ... `>>` *file_name*

will append the command's output to the file *file_name*.

6.11.1 `HIDEOUT`

Syntax classification: UNC

Syntax: `HIDEOUT` *file_name* [`APPEND`]

Purpose: Redirects all subsequent MERLIN output to the specified file. The optional argument `APPEND` causes `HIDEOUT` to append the output to the file, instead of overwriting it. The effect of a `HIDEOUT` command is canceled by `REVEAL`. Nested pairs of `HIDEOUT`–`REVEAL` commands are allowed.

6.11.2 `REVEAL`

Syntax classification: SIC

Purpose: Cancels the effect of the most recent `HIDEOUT` command.

6.12 Macro and MCL related commands

6.12.1 `MACRO`

Syntax classification: SIC

Purpose: `MACRO` is an interactive command, that assists the user in preparing a macro. It initiates a macro composing session that prompts for the macro name and the constituent commands, and finally appends the resulting macro in file `MACROF`. To terminate the macro composition one must use the `CLEAR` command.

Ind	Keyword	Description	Value	Allowed values
1)	DO	Desired action	w	{c,r,w}
2)	PROB	Confidence Probability	0.6827	0. < real < 1.
3)	FILE	File name	COVAR	Any string
4)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.21: The `COVARIANCE` panel.

6.12.2 `CLEAR`

Syntax classification: SIC

Purpose: Used to terminate a macro composing session initiated by command `MACRO`. It has no effect outside a `MACRO` session.

6.12.3 `RUNMCL`

Syntax classification: UNC

Syntax: `RUNMCL` *file_name*

Purpose: Initiates execution of a precompiled MCL program, that resides in the specified file.

Equivalent syntax: `-`*file_name*

6.13 Data fitting related commands

6.13.1 `COVARIANCE`

Syntax classification: PAC

Purpose: Calculates the covariance matrix for a least squares fit. The associated panel is listed in figure 6.21. The panel parameters are as follows:

`DO` Specifies the action to be taken. Allowed values are:

- `C` Calculate the covariance matrix.
- `R` Read the covariance matrix from a file.
- `W` Calculate and store the covariance matrix in a file.

`PROB` The probability that is used to define the confidence region.

`FILE` The file to store (or read from) the covariance matrix. The matrix is stored column wise and may be later used by the `CONFIDENCE` command.

`CANCEL` This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

6.13.2 `CONFIDENCE`

Syntax classification: RAC

Purpose: Displays the equation defining the confidence region for a least squares fit.

6.14 Getting help

There are two commands that provide useful information on the usage of MERLIN.

6.14.1 `LIST`

Syntax classification: SIC

Purpose: Lists alphabetically all MERLIN commands. The command's output is listed in figure 6.22.

6.14.2 `HELP`

Syntax classification: UNC

Syntax: `HELP` *comand_name*

Purpose: Provides on–line instructions for the usage of all MERLIN commands.

Examples: `HELP POINT` produces the output listed in figure 6.23.

```
--------------------------------------------------------------------------------
                             Merlin commands
--------------------------------------------------------------------------------
   ACCUM       ADJUST      ALIAS       ALIASDIS    ANAL        AUTO
   BACKUP      BATCH       BFGS        CFLAG       CFLAGDIS    CLEAR
   CONFIDENCE  CONGRA      CONTROL     COVARIANCE  DELETE      DFP
   DISCARD     DUMP        ECHO        EPILOG      EVALUATE    FAST
   FIX         FIXALL      FLAG        FLAGDIS     FULLBACK    FULLPRINT
   GENERAL     GNORM       GODFATHER   GOEOF       GRADCHECK   GRADDIS
   GRAPH       HALFPRINT   HELP        HESSIAN     HIDEOUT     HISTORY
   IAF         INIT        INSPECT     JANAL       JCOMPARE    JNUMER
   LASTBACK    LDEMARGIN   LEVE        LIMITS      LIST        LMARGIN
   LOOSALL     LOOSE       MACRO       MAD         MEMO        MIXED
   MODEDIS     NOBACK      NOEVAL      NONAME      NOPRINT     NOTARGET
   NUMER       PANELOFF    PANELON     PDUMP       PICK        POINT
   PSGRAPH     PSTATUS     QUAD        QUIT        RDEMARGIN   RESET
   RETURN      REVEAL      REWIND      RMARGIN     ROLL        RUNMCL
   SHORTDIS    SIMPLEX     SOS         STEP        STEPALL     STEPDIS
   STOP        TARGET      TERMDIS     TITLE       TOLMIN      TRUST
   UNALIAS     VALDIS

Use the HELP command to obtain more information.
```

Figure 6.22: Output from the LIST command.

```
--------------------------------------------------------------------------------
Syntax: POINT index_specification value ...

Description:
POINT assigns values to the minimization parameters.

MCL equivalent: POINT ( X.index=value [; X.index=value ... ] )

Examples: POINT 1 1.5 2 -3.65
          POINT 1-8 12.5

See also: SHORTDIS, INIT.
--------------------------------------------------------------------------------
```

Figure 6.23: Sample output from the HELP command.

```
tolmin <cr>
TOLMIN
------------------------------------------------------------------------------
                                 TOLMIN Panel
------------------------------------------------------------------------------
Ind Keyword    Description                    Value     Allowed values
 1) NOC        Number of calls                 300      Any int >= 1
 2) PRINT      Printout level                   1       {0,1,2}
 3) ACC        Termination accuracy           0.        0. <= real < 1.
 4) MAD        Automatic derivatives           Off      {On,Off}
 5) CANCEL     Cancel / Proceed                 1       {0,1}
------------------------------------------------------------------------------
Enter changes:
mad ? <cr>
------------------------------------------------------------------------------


MAD: Specifies whether the Merlin Automatic Derivatives are to be used
     when approximating the gradient vector. Allowed values are:
       ON   The Merlin Automatic Derivatives are to be used.
       OFF  Use the current derivative mode.
------------------------------------------------------------------------------
```

Figure 6.24: Requesting help on panel keywords.

6.14.3 Help on panel keywords

When a panel with several keywords appears, one can obtain information for their meaning on–line by entering a keyword (or an index) followed by a question mark. As an example we list the user's input and the corresponding output in figure 6.24. If a single question mark is entered at the panel's prompt, information for all the parameters is issued.

6.15 Odds and ends

6.15.1 ECHO

Syntax classification: UNC

Syntax: ECHO *any_message*

Purpose: Displays its arguments. It can be used inside macros, to issue informative messages.

Ind	Keyword	Description	Value	Allowed values
1)	DISKMAC	Macros on regular files	1	{0,1}
2)	ECHOMAC	Macro start/end messages	1	{0,1}
3)	ECHOPRO	Prolog/epilog messages	1	{0,1}
4)	MCLKEYS	List MCL keywords	0	{0,1}
5)	ECHOCOM	Echo commands	1	{0,1}
6)	KEEPIN	Keep input file	1	{0,1}
7)	KEEPOUT	Keep output file	1	{0,1}
8)	MCLBUF	Use buffered MCL	1	{0,1}

Figure 6.25: The `CONTROL` panel.

Examples: `ECHO ----- Minimization starts now !!!`
`ECHO 'A     very     long     message'`

6.15.2 EPILOG

Syntax classification: UNC

Syntax: `EPILOG` *epilog_command*

Purpose: Defines a command to be executed after a `STOP`, `RETURN` or `QUIT` is issued by the user. A blank argument (ie: `EPILOG ' '`) clears the epilog command. The current epilog can be displayed using `MODEDIS`.

6.15.3 CONTROL

Syntax classification: PAC

Purpose: Sets the parameters that control some Merlin features. The associated panel is listed in figure 6.25. The panel parameters are as follows:

`DISKMAC` Determines whether Merlin should check in plain disk files for macros not present in the macro file `MACROF`. For example if the macro `.abc` is not present in the macro file, Merlin would check for the existence of file `abc`. Allowed values are:

- 0 Do not check for macros on disk files.
- 1 Check for macros on disk files.

`ECHOMAC` This setting affects the printing of informative messages indicating that a macro has started / stopped executing. Allowed values are:

- 0 Do not issue informative messages.
- 1 Issue informative messages.

`ECHOPRO` Determines whether informative messages are printed each time the MERLIN prolog / epilog begins or ends execution. Allowed values are:

- 0 Do not issue informative messages.
- 1 Issue informative messages.

`MCLKEYS` Specifies whether the MCL specific keywords and their values (that are normally returned to an MCL program) should be printed after a panel command terminates. MCL specific keywords are identified in the panel description file by the trailing question mark, eg: `INFO?` Allowed values are:

- 0 Do not print the MCL specific keywords.
- 1 Print the MCL specific keywords.

`ECHOCOM` Controls printing of the current command before executing it. Allowed values are:

- 0 Do not print the name of current command.
- 1 Print the name of current command.

`KEEPIN` Controls the handling of the MERLIN input file when MERLIN execution ends. Allowed values are:

- 0 Delete the MERLIN input file when MERLIN execution ends.
- 1 Do not delete the MERLIN input file when MERLIN execution ends.

This setting only applies when the configuration directive `INPUT_FILE` has been used.

`KEEPOUT` Controls the handling of the MERLIN output file when MERLIN execution ends. Allowed values are:

- 0 Delete the MERLIN output file when MERLIN execution ends.
- 1 Do not delete the MERLIN output file when MERLIN execution ends.

This setting only applies when the configuration directive `OUTPUT_FILE` has been used.

`MCLBUF` Determines whether an MCL program should run in buffered or unbuffered mode. While in buffered mode, MERLIN will attempt to read in all of the object file specified in the `RUNMCL` command and keep it in memory. If the program is too large to fit in the buffer, MERLIN reverts to the unbuffered mode. In the later case, the `RUNMCL` command will inform the user. Allowed values are:

- 0 Run MCL programs unbuffered.
- 1 Use the buffer.

Ind	Keyword	Description	Value	Allowed values
1)	FILE	History file name	HISTORY	Any string
2)	STATUS	History status	Off	{On,Off}
3)	OLD	Old file (Append/Delete)	A	{A,D}
4)	CANCEL	Cancel / Proceed	1	{0,1}

Figure 6.26: The HISTORY panel.

6.15.4 HISTORY

Syntax classification: PAC

Purpose: Controls the MERLIN history mechanism. When history is active, all MERLIN commands are stored in a file. The file can be used as a macro to repeat a sequence of commands. The associated panel is listed in figure 6.26. The panel parameters are as follows:

FILE The file to store the MERLIN commands.

STATUS Determines the status of the history mechanism. Allowed values are:

- ON The history mechanism is turned on.
- OFF The history mechanism is turned off.

OLD Specifies what happens if the history mechanism is turned on, and the history file, exists already. Allowed values are:

- A Append to the already existing file.
- D Overwrite the previous contents of the history file.

CANCEL This parameter controls if the intended procedure will be executed or canceled. If any changes to any of the panel parameters have been entered they are recorded anyway.

- 1 is the default value and the process advances normally.
- 0 cancels the action initially intended.

Chapter 7

Extensions

7.1 Why extend

After working with MERLIN for a while, one realizes the need for incorporating his own code fragments into MERLIN. For example one may want to write to a file the resulting minimization parameters using a strange format, produce a graph for a plotter, or even add his own minimization routine. The possibilities are infinite. We decided to incorporate a mechanism for uniformly extending MERLIN, without having to modify the body of the program. Any extensions added, are automatically recognized by MCL programs as well. The procedure involves three steps:

- The user has to provide a subroutine that performs the desired operations. A user subroutine, that is used as a MERLIN extension, is termed a *plug–in module*.
- A unique name must be chosen for the plug–in. This name must be declared in the MERLIN configuration file.
- In case the plug–in obtains its input using the panel mechanism, one must make the corresponding entries in the panel description file.

These steps are further explained in the following sections.

7.2 Writing the plug–in module

Let us assume that one wants to create an alternative to `SHORTDIS`: a plug–in module that will print the value of the parameters, their names (if any) and their first partial derivatives. A subroutine that performs the desired operation is shown in figure 7.1. Sample output (for the two parameter Rosenbrock function) is demonstrated in figure 7.2. As is clear from the example, to

Source code	*Short description*
`SUBROUTINE NEWSH`	
`CHARACTER*10 NI`	
`CALL GETIOU(IUINP,IUOUT)`	Find out the output unit number (`IUOUT`)
`CALL GETDIM(N,M)`	Find out the number of parameters (`N`)
`DO 10,I=1,N`	Loop over all parameters
`CALL GETX1(I,XI)`	Get the value of the `I`$^{\text{th}}$ parameter (`XI`)
`CALL GETNM1(I,NI)`	Get the name of the `I`$^{\text{th}}$ parameter (`NI`)
`CALL GETG1(I,GI)`	Get the `I`$^{\text{th}}$ component of the gradient (`GI`)
`WRITE (IUOUT,20) I, NI, XI, GI`	Print them all out
`10 CONTINUE`	
`20 FORMAT (2X,I5,2X,A,2(2X,1PE14.7))`	
`END`	

Figure 7.1: Sample plug–in module: an alternative to `SHORTDIS`. It uses the glue routines `GETIOU`, `GETDIM`, `GETX1`, `GETNM1` and `GETG1`.

```
    1  Rho          4.0000000E+00   1.9206000E+04
    2  Sigma        4.0000000E+00  -2.4000000E+03
```

Figure 7.2: Output from the sample plug–in module `NEWSH`.

access information about the minimization parameters and the MERLIN operating environment we use a number of routines: `GETDIM`, `GETIOU`, `GETX1`, `GETNM1`, `GETG1`. These are termed *glue routines* and their purpose is to provide a standard programming interface to the internal MERLIN data structures. There are more glue routines; a complete list is given in chapter 9.

After writing the subroutine, one must insert a call to it in one of the empty plug–in subroutines provided in MERLIN. For example:

```
SUBROUTINE PLUG1
CALL NEWSH
END
```

There are 50 empty subroutines (`PLUG1` ... `PLUG50`) that can be used to accommodate plug–in modules. Naturally, the file containing `SUBROUTINE PLUG1` has to be recompiled.

7.3 Naming the plug–in module

Plug–in modules are referenced through a symbolic name chosen by the user. The name for the plug in module must be declared in the MERLIN configuration file using the `PLUG` configuration directive. It can be up to 10 characters long and must be unique among all MERLIN commands and other plug–in modules. For the above example we chose the name `NEWSH`. The corresponding entry in the configuration file should be:

```
PLUG 1 NEWSH
```

The plug–in can be invoked using the name `NEWSH`. In addition all other MERLIN features, such as command aliasing, output redirection etc., hold for the new plug–in as well. The new plug–in is also recognized by the MCL compiler.

7.4 Adding on–line help

The help texts for all non–panel commands reside in the MERLIN help file named `HELP`.[1] The structure of the help file is shown in figure 7.3. *help_text*$_1$, *help_text*$_2$, ... are multi line help texts that appear whenever requested by a `HELP` command. *initial_help_text* appears whenever a `HELP` command without any arguments is issued. Part of the help file that corresponds to the sample plug–in module `NEWSH` is shown in figure 7.4.

[1] The default name can be changed using the `HELP_FILE` configuration directive.

initial_help_text
***$command\_name_1$
$help\_text_1$
***$command\_name_2$
$help\_text_2$
...

Figure 7.3: The structure of the MERLIN help file.

```
***NEWSH
NEWSH is a sample plug-in module that demonstrates the use of
the Merlin glue routines. It does not require any command line
arguments and its output resembles the SHORTDIS command.
```

Figure 7.4: Help text for the sample plug–in module `NEWSH`.

7.5 Plug–ins with command line arguments

Plug–in modules can take advantage of any arguments in the MERLIN command line. A more elaborate example is presented in figure 7.5: a plug–in module that will write the value of the parameters, their names (if any) and their first partial derivatives to a user specified file. The file name will be given as a command line argument. If we use the second empty plug–in subroutine:

```
SUBROUTINE PLUG2
CALL FILESH
END
```

and choose the name `FILEDIS`, then an appropriate entry in the configuration file would be:

```
PLUG 2 FILEDIS
```

The plug–in module should then be invoked with exactly one argument, the file name:

`FILEDIS` *some_file*

7.6 Plug–ins with a panel

Plug–in modules can use the panel mechanism to obtain input and to communicate values to MCL programs. Figure 7.6 presents a trivial example of a plug–in module that writes some of

Source code	*Short description*
`SUBROUTINE FILESH`	
`CHARACTER*10 NI`	
`CHARACTER*80 FNAME`	
`CALL GETIOU(IUINP,IUOUT)`	Find out the output unit number (`IUOUT`)
`CALL ARGNO(NARG)`	Find out number of arguments (`NARG`)
`IF (NARG.NE.1) THEN`	If incorrect number of arguments
`   WRITE (IUOUT,*) 'One arg please'`	
`   CALL SETCOD(-2)`	Indicate an error
`   RETURN`	
`END IF`	
`CALL GETARG(1,FNAME,LENF)`	Get the first argument
`NU = NUNIT()`	Request a free unit number
`OPEN (NU,FILE=FNAME)`	Open the file
`CALL GETDIM(N,M)`	Find out the number of parameters (`N`)
`DO 10,I=1,N`	Loop over all parameters
`   CALL GETX1(I,XI)`	Get the value of the `I`th parameter (`XI`)
`   CALL GETNM1(I,NI)`	Get the name of the `I`th parameter (`NI`)
`   CALL GETG1(I,GI)`	Get the `I`th component of the gradient (`GI`)
`   WRITE (NU,20) I, NI, XI, GI`	Print them all out
`10 CONTINUE`	
`CLOSE (NU)`	Close the file
`CALL SETCOD(0)`	Indicate no error
`20 FORMAT (2X,I5,2X,A,2(2X,1PE14.7))`	
`END`	

Figure 7.5: Sample plug–in module that takes advantage of command line arguments. It uses the glue routines `GETIOU`, `ARGNO`, `SETCOD`, `GETARG`, `NUNIT`, `GETDIM`, `GETX1`, `GETNM1` and `GETG1`.

the minimization parameters to a file. In addition it can return to a requesting MCL program the number of lines written to the file. The third empty plug–in subroutine is used for this example:

```
SUBROUTINE PLUG3
CALL PEXA
END
```

The name of the plug–in (PEXA) is declared in the configuration file as:

```
PLUG 3 PEXA
```

The PEXA panel as presented by MERLIN when the command is invoked, is shown in figure 7.8. The panel uses three keywords: FROM, TO and FILE. The following section describes the format of the panel description file and explains how to make the appropriate entries. Part of the panel description file that correspond to the PEXA sample plug–in, is shown in figure 7.7.

7.7 The panel description file

All MERLIN panels and their corresponding keywords along with their default values are stored in the panel description file. Its default name is PDESC and can be changed through the configuration directive PDESC_FILE. MERLIN commands and user plug–in modules share the same panel description file. Note that the panel description file *must* be present in order for MERLIN to operate.

The format of the panel description file is explained in the following sections. Lines starting with % (comments) and blank lines are ignored.

7.7.1 Declaring the panel name

The description of a panel starts with a `.panel` line, and extends until another `.panel` line or the end of the description file is encountered. The format of a `.panel` line is:

`.panel` *panel_name* *panel_status*

panel_name — The MERLIN command that this panel corresponds to. In case of a plug–in module, it is the name assigned to the plug–in through the PLUG configuration directive. It must be up to 10 characters long, and is case insensitive.

panel_status — Initial setting for the panel status. It can be either `on` or `off`. The panel status is changed by the PANELON and PANELOFF commands.

Examples:
```
.panel bfgs on
.panel memo off
```

Source code	*Short description*
`SUBROUTINE PEXA`	
`CHARACTER*80 FNAME`	
`CALL CHANGE(MERR)`	Present the panel and accept changes
`IF (MERR.NE.0) THEN`	Check if any error occurred
`   CALL SETCOD(MERR)`	Indicate an error
`   RETURN`	
`END IF`	
`CALL GETPI('FROM',NFROM)`	Get index to start from (`NFROM`)
`CALL GETPI('TO',NTO)`	Get index to end to (`NTO`)
`CALL GETPS('FILE',FNAME,LEF)`	Get the file name (`FNAME`)
`NU = NUNIT()`	Obtain a free unit number (`NU`)
`OPEN (NU,FILE=FNAME)`	Open the file
`DO 10,I=NFROM,NTO`	Loop over the specified parameters
`   CALL GETX1(I,XI)`	Get the `I`th parameter (`XI`)
`   WRITE (NU,*) XI`	Write the parameter
`10 CONTINUE`	
`CLOSE (NU)`	Close the file
`NL = NTO-NFROM+1`	Calculate the number of lines written (`NL`)
`CALL SETPI('LINES',NL,IERR)`	Return the number of lines written
`CALL SETCOD(0)`	Indicate no error
`END`	

Figure 7.6: Sample plug–in module that exploits the panel mechanism. It uses the glue routines `CHANGE`, `SETCOD`, `GETPI`, `GETPS`, `NUNIT`, `GETX1` and `SETPI`.

```
.PANEL PEXA ON
FROM     I   [1,inf)        1       'Index to begin from'
TO       I   [1,inf)        1       'Index to end to'
FILE     S   {any}        'SAMPLE'  'File to write to'
LINES?   I   [1,inf)        1       'Lines written'

.HELP
The PEXA plug-in module disposes some of the minimization parameters
in a file.
.END
.HELP FROM
FROM specifies the index to start writing from.
.END
.HELP TO
TO specifies the index to end writing to.
.END
.HELP FILE
FILE specifies the file to write to.
.END
```

Figure 7.7: Part of the panel description file corresponding to the `PEXA` sample plug–in module.

```
--------------------------------------------------------------------------------
                                  PEXA Panel
--------------------------------------------------------------------------------
Ind Keyword    Description                    Value     Allowed value
 1) FROM       Index to begin from              1       Any int >= 1
 2) TO         Index to end to                  1       Any int >= 1
 3) FILE       File to write to              SAMPLE     Any string
```

Figure 7.8: The panel of the `PEXA` sample plug–in module as presented by MERLIN.

7.7.2 Declaring the keywords

After declaring the panel name, all keywords that appear in the panel must be described. If necessary, the description of a keyword can be split across several physical lines using the continuation character `&` at the end of each continued line. The description of a keyword has the form:

keyword keyword_type allowed_values default_value short_description

keyword — The keyword name. It must be up to 10 characters long and is case insensitive. It must be unique in the current panel. Keywords of type `I` or `R` may be followed by a question mark to designated an MCL returned keyword. The values of MCL returned keywords, are set by the plug–in module and are accessible by an MCL program.

keyword_type — A single character indicating the keyword type, i.e. what kind of values are associated with the keyword. Possible keyword types are:

- `I` The keyword is associated with integer numbers.
- `R` The keyword is associated with floating point numbers.
- `C` The keyword is associated with character strings. MERLIN uses case insensitive comparisons to determine whether an allowed value has been assigned to the keyword.
- `S` The keyword is associated with character strings. MERLIN uses case sensitive comparisons to determine whether an allowed value has been assigned to the keyword.

allowed_values — A description of the values allowed for the keyword in one of the following forms:

- For keywords of type `I` or `R` an interval of allowed values in the form:

  ls ll , *ul rs*

  ls is either `(` for an open interval or `[` for a closed interval.

  rs is either `)` for an open interval or `]` for a closed interval.

  ll, ul The lower and upper limits of the interval correspondingly. They must be integer or floating point numbers according to the corresponding keyword type. The symbols `inf` and `-inf` can be used to designate plus or minus infinity. Note that always $ll \leq ul$.

  Examples: `(0,1)` `[0,inf)` `(-inf,inf)` `(0.5,2.5]` `[-10,10]`

- A set of allowed values in the form:

{ $value_1$, $value_2$, ... $value_k$ }

The values in the set must be integer or floating point numbers, or character strings (enclosed in quotes), according to the corresponding keyword type. In the case of keywords of type `C` or `S`, the special entry `{any}` will cause any value to be accepted.

Examples: `{1,2,3}` `{any}` `{'a','b','c'}` `{'on','off'}`

default_value The default value for the keyword. It should be one of the allowed values.

short_description A short description of the keyword that will appear in the panel. It must be enclosed in a pair of single quotes.

Examples:

```
NOC    I  [1,inf)     300       'Number of calls'
PRINT  I  {0,1,2}      1        'Printout level'
FILE   S  {any}     'HESSIAN'   'File to write to'
WHAT   C  {'F','D'}   'F'       'What to plot (Fun/Data)'
INFO?  I  [0,inf)      0        'Result code'
```

7.7.3 Adding help texts

An explanatory help text for the command itself and for each of the keywords can be included in the panel description file. The help text for the command itself is displayed using the MERLIN `HELP` command:

`HELP` *command*

The help text for a specific keyword is displayed using a question mark after its name:

command keyword `?`

Help texts for all panel commands as well for all of their keywords is included in the distributed panel description file. Note that the help texts for all non–panel commands reside in the MERLIN help file. The help texts in the panel description file have the form:

`.help` [*keyword*]
 help_text
`.end`

keyword The name of a keyword which if present associates the help text with the keyword. If *keyword* is omitted, the help text is associated with the command itself.

help_text A multi line help text. All lines in between `.help` and `.end`, including blank lines and lines starting with the percent character, are considered as part of the help text.

Chapter 8

The Merlin–Mcl configuration file

8.1 General description

The purpose of the configuration file is to set system dependent parameters (such as the largest floating point number) and control the various defaults (such as the printout level, processing mode, etc.) in a portable way. The configuration file is read by both, Merlin and Mcl and its default name `CONFIG` can be changed during installation.

Each line of the configuration file contains a configuration directive and optionally one or more parameters:

directive parameter$_1$ *parameter*$_2$ . . . *parameter*$_k$

Lines starting with the percent character (comments) and blank lines are ignored. Directives and parameters are case insensitive, with the exception of filenames, which must adhere to the local operating system conventions and can be up to 80 characters long. Arguments with space or tab characters must be enclosed in a pair of single quotes. A quote is generated using the escape sequence `\'`. Leading and trailing spaces and tabs are ignored outside a pair of single quotes. In case of multiple instances of a directive, the last one takes effect. `PLUG` is an exception, where multiple `PLUG` directives are used to define various user plug–in modules.

If an error occurs while reading the configuration file, Mcl will print appropriate messages and abort execution. Merlin will return control to the calling program, indicating the problem through the return flag `IQUIT`. Possible values are:

`IQUIT` $= -3$ Some errors occurred while parsing the configuration file. Moreover the Merlin output file could not be opened.

`IQUIT` $= -4$ Some error occurred while parsing the configuration file. The error is related to the directives OUTPUT_FILE, OUTPUT_UNIT or OUTPUT_PRECONN that control the output file.

IQUIT = −5 The configuration file has been successfully parsed. The output file could not be opened however.

IQUIT = −6 Some error occurred while parsing the configuration file. The error is not related to the directives OUTPUT_FILE, OUTPUT_UNIT or OUTPUT_PRECONN that control the output file.

Table 8.1 summarizes the defaults for all configuration directives in case some of them are omitted, or a configuration file is not used.

8.2 Directives that control MERLIN input–output

8.2.1 MERLIN input–output units and files

Usually a Fortran statement of the form

```
READ (*,*) x
```

expects its input from the keyboard. Likewise the statement

```
WRITE (*,*) x
```

displays its output to the screen of a terminal. Fortran compilers have a pair of unit numbers (for example 5 and 6) that refer to the keyboard and terminal screen correspondingly and accomplish the same effect. These are referred to as the *standard input and output unit numbers* correspondingly. For example:

```
READ (5,*) x
WRITE (6,*) x
```

Some operating systems use unit numbers other than 5 and 6, or allow the user to assign its own unit numbers to the input–output devices. Some others require a Fortran OPEN statement before the device can be used for input–output operations. In addition, one may want to use regular disk files for input and output instead of the usual devices. This section describes the configuration directives that control the way MERLIN handles its input and output.

8.2.2 INPUT_FILE

Syntax: INPUT_FILE *file_name*

Configuration directive	*Default value*
INPUT_FILE	None.
OUTPUT_FILE	None.
INPUT_UNIT	5
OUTPUT_UNIT	6
INPUT_PRECONN	yes
OUTPUT_PRECONN	yes
HEADER	on
PRINTOUT	full
PDESC_FILE	PDESC
MACRO_FILE	MACROF
HELP_FILE	HELP
MCL_ERROR_FILE	ERROR
MCL_OBJECT_FILE	The standard output unit
HAS_APPEND	no
SIZE_REAL	4
SIZE_INT	4
SIZE_CHAR	1
UNIT_RANGE	50 80
ONEOF	return
FILE	Not applicable
MODE	iaf
PROLOG	No prolog is executed
EPILOG	No epilog is executed
PLUG	Not applicable
BIGGER	10^{36}
SMALLER	10^{-36}
MACHINE_DIGITS	As determined at MERLIN startup

Table 8.1: Default values for all configuration directives.

Purpose: `INPUT_FILE` specifies that MERLIN should read its input from *file_name* instead of the standard input unit.

Default: None. If this directive is omitted, MERLIN reads its input from the standard input unit.

Used by: MERLIN only.

Examples:
```
INPUT_FILE test
INPUT_FILE 'sample input'
INPUT_FILE c:\merlin\in\sample.in
```

8.2.3 OUTPUT_FILE

Syntax: `OUTPUT_FILE` *file_name* [`APPEND`]

Purpose: `OUTPUT_FILE` specifies that MERLIN should dispose its output to *file_name* instead of the standard output unit. If the optional argument `APPEND` is present and *file_name* exists already, output will be appended at the end of the file.

Default: None. If this directive is omitted, MERLIN disposes its output to the standard output unit.

Used by: MERLIN only.

Examples:
```
OUTPUT_FILE fit.out append
OUTPUT_FILE ..\results\run01
```

8.2.4 INPUT_UNIT

Syntax: `INPUT_UNIT` *unit_number*

Purpose: `INPUT_UNIT` specifies the Fortran unit number that refers to the standard input device (keyboard), i.e. a Fortran statement of the form

`READ (`*unit_number*`,*) x`

should be able to receive input from the keyboard.

Default: If this directive is omitted, *unit_number* defaults to 5. This value is suitable for most compilers running under Unix.

Used by: MERLIN only.

Examples:
```
INPUT_UNIT 9
```

8.2.5 OUTPUT_UNIT

Syntax: `OUTPUT_UNIT` *unit_number*

Purpose: `OUTPUT_UNIT` specifies the Fortran unit number that refers to the standard output device (terminal), i.e. a Fortran statement of the form

`WRITE (`*unit_number*`,*) x`

should be able to write output to the terminal.

Default: If this directive is omitted, *unit_number* defaults to 6. This value is suitable for most compilers running under Unix.

Used by: MERLIN only.

Examples: `OUTPUT_UNIT 9`

8.2.6 INPUT_PRECONN

Syntax: `INPUT_PRECONN` *preconnect*

Purpose: Specifies whether the Fortran input unit, as specified by the `INPUT_UNIT` directive, is preconnected to the corresponding input device. Preconnected input units need no Fortran `OPEN` statements before they can be used for input. *preconnect* may be either `yes` or `no`.

Default: If this directive is omitted, `INPUT_PRECONN yes` is assumed.

Used by: MERLIN only.

Examples: `INPUT_PRECONN yes`

8.2.7 OUTPUT_PRECONN

Syntax: `OUTPUT_PRECONN` *preconnect*

Purpose: Specifies whether the Fortran output unit, as specified by the `OUTPUT_UNIT` directive, is preconnected to the corresponding output device. Preconnected output units need no Fortran `OPEN` statements before they can be used for output. *preconnect* may be either `yes` or `no`.

Default: If this directive is omitted, `OUTPUT_PRECONN yes` is assumed.

Used by: MERLIN only.

Examples: `OUTPUT_PRECONN yes`

8.2.8 HEADER

Syntax: `HEADER` *header_status*

Purpose: Determines whether the informative messages that are normally printed during MERLIN startup will be issued. The messages involve version of the program, authors and e-mail contact. *header_status* can be either `on` or `off`.

Default: `HEADER on`, i.e. messages will be printed.

Used by: MERLIN only.

Examples: `HEADER off`

8.2.9 PRINTOUT

Syntax: `PRINTOUT` *printout_level*

Purpose: Specifies the amount of output that will be issued, while MERLIN starts, and until the MERLIN prompt is reached for the first time. This setting also affects the output of the prologue command. *printout_level* can be `full`, `half` or `no`. The three possible arguments are analogous to the MERLIN commands `FULLPRINT`, `HALFPRINT` and `NOPRINT` that change the printout mode after MERLIN has started. `PRINTOUT no` suppresses all output. With `PRINTOUT half` only error messages are issued. `PRINTOUT full` allows all output to be printed.

Default: If this directive is omitted, `PRINTOUT full` is assumed.

Used by: MERLIN only.

Examples: `PRINTOUT half`

8.3 File related directives

8.3.1 PDESC_FILE

Syntax: `PDESC_FILE` *file_name*

Purpose: Sets the name of the MERLIN panel description file to *file_name*.

Default: If this directive is omitted, the name of the MERLIN panel description file defaults to `PDESC`.

Used by: MERLIN and MCL.

Examples: `PDESC_FILE enhanced-PDESC`
`PDESC_FILE /usr/local/merlin/pdesc`

8.3.2 MACRO_FILE

Syntax: `MACRO_FILE` *file_name*

Purpose: Sets the name of the MERLIN macro file to *file_name*.

Default: If this directive is omitted, the name of the MERLIN macro file defaults to `MACROF`.

Used by: MERLIN only.

Examples: `MACRO_FILE macro1`
`MACRO_FILE HD:merlin:supermac`

8.3.3 HELP_FILE

Syntax: `HELP_FILE` *file_name*

Purpose: Sets the name of the MERLIN help file to *file_name*.

Default: If this directive is omitted, the name of the MERLIN help file defaults to `HELP`.

Used by: MERLIN only.

Examples: `HELP_FILE extrahelp`

8.3.4 MCL_ERROR_FILE

Syntax: `MCL_ERROR_FILE` *file_name*

Purpose: Sets the name of the default MCL error file to *file_name*. When the an error file (`E=`*file_name*) is not supplied when the MCL compiler is invoked, the error file defaults to this value. Setting *file_name* to an empty string (`MCL_ERROR_FILE ' '`) causes all compiler diagnostics to be printed on the standard output unit.

Default: If this directive is omitted, all compiler diagnostics are printed on the standard output unit.

Used by: MCL only.

Examples: `MCL_ERROR_FILE errors`
`MCL_ERROR_FILE errs/job1`

8.3.5 MCL_OBJECT_FILE

Syntax: `MCL_OBJECT_FILE` *file_name*

Purpose: Sets the name of the default MCL object file to *file_name*. When the an object file (`B=`*file_name*) is not supplied when the MCL compiler is invoked, the object file defaults to this value.

Default: if this directive is omitted, `MCL_OBJECT_FILE` defaults to `MOC` (standing for MERLIN Object Code).

Used by: MCL only.

Examples: `MCL_OBJECT_FILE myprog`

8.3.6 HAS_APPEND

Syntax: `HAS_APPEND` *append_status*

Purpose: Specifies whether your Fortran compiler supports the `APPEND` extension in an `OPEN` statement. More specifically, whether the following is a valid Fortran statement

`OPEN (UNIT=`*unit_number*`, FILE=`*file_name*`, ACCESS='APPEND')`

The `APPEND` extension allows much faster updating of large text files, eg those produced by the MERLIN backup mechanism, or by the `MEMO` command. *append_status* cane be either `yes` or `no`.

Default: If this directive is omitted, it defaults to `HAS_APPEND no`, i.e. the `APPEND` extension is not used.

Used by: MERLIN only.

Examples: `HAS_APPEND yes`

8.3.7 SIZE_REAL

Syntax: `SIZE_REAL` n

Purpose: Specifies that a *floating point* Fortran variable occupies n storage locations in an unformatted file. *Floating point* refers to a real or double precision variable, depending on the installation. n is used to calculate the appropriate `RECL` specifier when opening a MERLIN binary file

`OPEN (UNIT=`*unit_number*`, FILE=`*file_name*`, RECL=`*record_length*`)`

Default: If this directive is omitted `SIZE_REAL` defaults to 4, which is appropriate for double precision variables in most 32 bit computer systems.

Used by: MERLIN only.

Examples: `SIZE_REAL 4`

8.3.8 SIZE_INT

Syntax: `SIZE_INT` *n*

Purpose: Specifies that an integer Fortran variable occupies *n* storage locations in an unformatted file. *n* is used to calculate the appropriate `RECL` specifier when opening a MERLIN binary file

`OPEN (UNIT=`*unit_number*`, FILE=`*file_name*`, RECL=`*record_length*`)`

Default: If this directive is omitted `SIZE_INTEGER` defaults to 4.

Used by: MERLIN only.

Examples: `SIZE_INTEGER 2`

8.3.9 SIZE_CHAR

Syntax: `SIZE_CHAR` *n*

Purpose: Specifies that a `CHARACTER*1` Fortran variable occupies *n* storage locations in an unformatted file. *n* is used to calculate the appropriate `RECL` specifier when opening a MERLIN binary file

`OPEN (UNIT=`*unit_number*`, FILE=`*file_name*`, RECL=`*record_length*`)`

Default: If this directive is omitted `SIZE_CHAR` defaults to 1.

Used by: MERLIN only.

Examples: `SIZE_CHAR 1`

8.3.10 UNIT_RANGE

Syntax: `UNIT_RANGE` *from to*

Purpose: Defines an allowed range of Fortran unit numbers to be used by MERLIN. The purpose of the unit range is to prevent MERLIN from interfering with files already open at the timeSUBROUTINE MERLIN is called, or with files used within the user written modules (FUNMIN, GRANAL, etc.).

Default: If this directive is omitted it defaults to UNIT_RANGE 50 80.

Used by: MERLIN only.

Examples: `UNIT_RANGE 25 40`

8.3.11 ONEOF

Syntax: ONEOF *action*

Purpose: Determines the action MERLIN should take when an end–of–file condition occurs in the input file. The specified action is taken only when MERLIN is running in interactive mode. When in batch mode, MERLIN always returns to the calling program, setting the IQUIT return flag to −10. *action* can be:

- **rewind** MERLIN rewinds and re–reads the input file.
- **ignore** MERLIN ignores the end–of–file condition and continues reading from the input file.
- **return** MERLIN immediately returns to the calling program, setting the IQUIT return flag to −10.

Default: If this directive is omitted it defaults to ONEOF return.

Used by: MERLIN only.

Examples: `ONEOF ignore`

8.3.12 FILE

Syntax: FILE *file_name*
file_contents
...END

Purpose: Allows the contents of an arbitrary file to be included in the configuration file. Upon MERLIN startup, all lines in between FILE and ...END are copied in the file *file_name*. If *file_name* exists already, its contents are overwritten. This directive can be used in conjunction with the PROLOG directive.

Default: Three is no default value. If this directive is omitted, no file is created.

Used by: MERLIN and MCL.

Examples:
```
FILE prl
POINT 1-8 4
SHORTDIS
...END
```

8.4 Miscellaneous directives

8.4.1 MODE

Syntax: `MODE` *processing_mode*

Purpose: Specifies that MERLIN should start in interactive (`iaf`) or batch (`batch`) processing mode. *processing_mode* can be either `iaf` or `batch`. The processing mode can be changed by the `IAF` and `BATCH` commands.

Default: If this directive is omitted it defaults to `MODE iaf`.

Used by: MERLIN only.

Examples: `MODE batch`

8.4.2 PROLOG

Syntax: `PROLOG` *prolog_command*

Purpose: Specifies that *prolg_command* is a command to be executed after the configuration file has been read; however before the first MERLIN prompt is displayed. *prolog_command* is any valid MERLIN command, macro or MCL program invocation. Additional arguments to the *prolog_command* are allowed as if it were issued interactively. As always the use of single quotes is mandatory, if spaces are to be embedded. Note that only one prolog command can be defined. If `PROLOG` is used more than once, the last one takes effect.

Default: None. If this directive is omitted, no prolog is executed.

Used by: MERLIN only.

Examples:
```
PROLOG 'point 1- 5'
PROLOG '.prl > prl.out'
PROLOG -an_mcl_prog
PROLOG .ini
```

8.4.3 EPILOG

Syntax: `EPILOG` *epilog_command*

Purpose: Specifies that *epilog_command* is a command to be executed, before MERLIN relinquishes control and either returns to the calling program (as a result of a `RETURN` or `QUIT` command), or stops (as a result of a `STOP` command). *epilog_command* is any valid MERLIN command, macro or MCL program invocation. Additional arguments to the *epilog_command* are allowed as if it were issued interactively. As always the use of single quotes is mandatory, if spaces are to be embedded. Note that only one prolog command can be defined. After MERLIN starts, the *epilog_command* can be changed, using the `EPILOG` command.

Default: None. If this directive is omitted, no prolog is executed.

Used by: MERLIN only.

Examples:
```
EPILOG shortdis
EPILOG 'dump what x file results'
```

8.4.4 PLUG

Syntax: `PLUG` n *plug_name*

Purpose: Defines the names for user written plug–in modules. More specifically it defines that *plug_name* should be used to reference the n^{th} plug–in. This directive should be used when the user has programmed `SUBROUTINE PLUG`n. Note that `plug_name` should not coincide with any of the MERLIN commands. In case one wants to reference the plug–in from an MCL program, the *plug_name* should not coincide with any of the MCL statements. (Including non–executable statements such as `PROGRAM`, `LOOP`, `VAR`, etc.)

Default: None. If this directive is omitted, no plug–in modules can be referenced, even if the are embedded in the source code.

Used by: MERLIN and MCL.

Examples:
```
PLUG 1 newsh
PLUG 7 supermin
```

8.4.5 BIGGER

Syntax: `BIGGER` β

Purpose: Specifies that β is approximately the largest floating point number your machine can handle. Note that according to installation, β refers to either real, or double precision arithmetic.

Default: If this directive is omitted, β defaults to 10^{36}, which is suitable for most computers in single precision.

Used by: MERLIN only.

Examples: `BIGGER 1.E+300`

8.4.6 SMALLER

Syntax: `SMALLER` σ

Purpose: Specifies that σ is approximately the smallest positive floating point number distinguishable from zero. Note that according to installation, σ refers to either real, or double precision arithmetic.

Default: If this directive is omitted, σ defaults to 10^{-36}, which is suitable for most computers in single precision.

Used by: MERLIN only.

Examples: `SMALLER 1.E-300`

8.4.7 MACHINE_DIGITS

Syntax: `MACHINE_DIGITS` n

Purpose: Specifies that during floating point operation approximately n significant digits are taken into account. Note that according to installation, n refers to either real, or double precision arithmetic.

Default: The default value is determined and printed at MERLIN startup. Use this directive if for some reason, MERLIN determines incorrectly the machine precision.

Used by: MERLIN only.

Examples:
```
MACHINE_DIGITS 7
MACHINE_DIGITS 15
```

Chapter 9

MERLIN glue routines

The following is a complete list of all MERLIN glue routines. Their purpose is to provide a standard programming interface to the internal MERLIN data structures. By using the glue routines instead of directly accessing the internal data structures, one avoids any knowledge of the internals of the MERLIN source. Furthermore he ensures compatibility of any plug–in modules with future MERLIN versions since the internal structures may change; the glue routines will not.

Most of the glue routines can be roughly classified as '*get*' or '*set*' routines. *Get* routines return information from the MERLIN run–time environment. *Set* routines alter the MERLIN run–time environment. For convenience the glue routines are divided in the following four categories:

- *Parameter related*: Glue routines that set or inquire data structures related to the minimization parameters.
- *Panel related*: Glue routines that manipulate the panel data structures.
- *Utility*: Routines that are not specific to MERLIN; they are included here however as a programming aid.
- *Miscellaneous*: The rest of the glue routines, that do not fall in one of the above categories.

In the description that follows, subprogram arguments are classified as:

- *Input* arguments: An initial value *must* be supplied when the glue routine is called. This value remains intact during operation of the routine.
- *Output* arguments: An initial value need not be supplied when the glue routine is called. These arguments will be assigned a value by the glue routine which will be returned to the calling program.

- *Input–output* arguments: An initial value *must* be supplied when the glue routine is called. These arguments will be assigned new values by the glue routine which will be returned to the calling program.

For efficiency reasons, there is only a minimal validity check on the arguments. Note that `REAL` variables and arrays should be replaced by `DOUBLE PRECISION` ones, for a double precision installation.

9.1 Parameter related glue routines

9.1.1 SUBROUTINE GETX

Definition:
```
SUBROUTINE GETX ( X )
REAL X(*)
```

Purpose: Returns the current values of all the minimization parameters.

Arguments:

- `X` input–output
  The current values of the minimization parameters with `X`(i) corresponding to parameter x_i. Array `X` should have at least N storage locations available.

9.1.2 SUBROUTINE GETX1

Definition:
```
SUBROUTINE GETX1 ( I, XI )
INTEGER I
REAL XI
```

Purpose: Returns the current value of one of the minimization parameters.

Arguments:

- `I` input
  Index of the minimization parameter to be returned.
- `XI` output
  The value of the `I`$^{\text{th}}$ minimization parameter.

9.1.3 SUBROUTINE SETX

Definition:
```
SUBROUTINE SETX ( X )
REAL X(*)
```

Purpose: Sets the current values of all the minimization parameters. This routine should always be used in conjunction with `SUBROUTINE SETVAL` so that the current value of the objective function corresponds to the current values of the minimization parameters.

Arguments:
- `X` input
  The values to set to the minimization parameters, with `X(`i`)` corresponding to x_i. The values `X(`i`)` should lie within their corresponding bounds. Moreover you are not allowed to assign a value different than the current one to a fixed parameter.

9.1.4 SUBROUTINE SETX1

Definition:
```
SUBROUTINE SETX1 ( I, XI )
INTEGER I
REAL XI
```

Purpose: Sets the current value of one of the minimization parameters.

Arguments:
- `I` input
  Index of the parameter to set.
- `XI` input
  The value to set to parameter x_{I}. It must lie within its corresponding bounds. Moreover you are not allowed to assign a value different than the current one if parameter x_{I} is fixed.

9.1.5 SUBROUTINE GETG

Definition:
```
SUBROUTINE GETG ( GRAD )
REAL GRAD(*)
```

Purpose: Returns the gradient vector, evaluated at the current point, using the derivative mode currently in effect.

Arguments:
- `GRAD` output
  The gradient vector with `GRAD(`i`)` corresponding to $\frac{\partial f}{\partial x_i}$. Array `GRAD` should have at least N storage locations available. `GRAD(`i`)` is set to zero if parameter x_i is fixed.

9.1.6 SUBROUTINE GETG1

Definition:
```
SUBROUTINE GETG1 ( I, GI )
INTEGER I
REAL GI
```

Purpose: Returns one of the components of the gradient vector evaluated at the current point, using the derivative mode currently in effect.

Arguments:

- `I` input
  Index of the parameter whose derivative is to be returned.
- `GI` output
  The first partial derivative with respect to parameter x_{I}. If parameter x_{I} is fixed, `GI` is set to zero.

9.1.7 SUBROUTINE GETHES

Definition:

```
SUBROUTINE GETHES ( H, LD, HOW )
REAL H(LD,*)
INTEGER LD
CHARACTER HOW
```

Purpose: Returns the Hessian matrix.

Arguments:

- `H` output
  The lower triangular part of `H` contains the second partial derivatives:

  $$\mathtt{H}(i,j) = \frac{\partial^2 f}{\partial x_i \partial x_j} \qquad i = 1 \ldots N, \quad j = 1 \ldots i$$

  The contents of the rest of the matrix are undefined. Matrix `H` should have at least N^2 storage locations available.
- `LD` input
  The leading dimension of matrix `H` with `LD` $\geq N$.
- `HOW` input
  Determines how the Hessian matrix is to be calculated. Allowed values are:
  - `HOW = 'F'` or `HOW = 'f'`
    Use only function values to approximate the Hessian matrix elements.
  - `HOW = 'G'` or `HOW = 'g'`
    Use only gradient values to approximate the Hessian matrix elements. This value should be used only if `SUBROUTINE GRANAL` is available.
  - `HOW = 'A'` or `HOW = 'a'`
    Call the user supplied `SUBROUTINE HANAL`.

9.1.8 SUBROUTINE GETJAC

Definition:
```
SUBROUTINE GETJAC ( FJ, LD )
REAL FJ(LD,*)
INTEGER LD
```

Purpose: Returns the Jacobian matrix, evaluated at the current point. The current Jacobian mode, as defined by the `JNUMER` or `JANAL` commands is used.

Arguments:

- `FJ` output
  The Jacobian matrix stored as:
  $$\texttt{FJ}(i,j) = \frac{\partial f_i}{\partial x_j}$$
  Matrix `FJ` should have at least $M \times N$ storage locations available.
- `LD` input
  The leading dimension of matrix `FJ` with `LD` $\geq M$.

9.1.9 SUBROUTINE GETMAR

Definition:
```
SUBROUTINE GETMAR ( MF, XL, XU )
DIMENSION MF(*)
REAL XL(*), XU(*)
```

Purpose: Returns the current upper and lower bounds for all minimization parameters. Bounds are normally set by the `LMARGIN` and `RMARGIN` commands and reset by the `LDEMARGIN` and `RDEMARGIN` commands. Each of the arrays `MF`, `XL` and `XR` should have at least N storage locations available.

Arguments:

- `MF` output
  For each minimization parameter x_i, `MF`(i) indicates whether an upper, lower, or both bounds have been set. Possible values are:
  - `MF`$(i) = 0$
    Neither an upper, nor a lower bound has been set for parameter x_i.
  - `MF`$(i) = 1$
    An upper bound has been set for parameter x_i.
  - `MF`$(i) = -1$
    A lower bound has been set for parameter x_i.
  - `MF`$(i) = 2$
    Both, an upper and a lower bound have been set for parameter x_i.

- `XL` output
  `XL(`i`)` stores the lower bound on parameter x_i. If x_i has no lower bound, a large negative number is returned.
- `XU` output
  `XU(`i`)` stores the upper bound on parameter x_i. If x_i has no upper bound, a large positive number is returned.

9.1.10 SUBROUTINE GETMR1

Definition:
```
SUBROUTINE GETMR1 ( I, MFI, XLI, XUI )
INTEGER I, MFI
REAL XLI, XUI
```

Purpose: Returns the upper and lower bound for one of the minimization parameters.

Arguments:
- `I` input
  Index of the parameter whose bounds are requested.
- `MFI` output
  Indicates whether the specified parameter has a lower, upper or both bounds. Possible values are:
  - `MFI = 0`
    Neither an upper, nor a lower bound has been set for parameter x_{I}.
  - `MFI = 1`
    An upper bound has been set for parameter x_{I}.
  - `MFI =` -1
    A lower bound has been set for parameter x_{I}.
  - `MFI = 2`
    Both, an upper and a lower bound have been set for parameter x_{I}.
- `XLI` output
  The lower bound. If a lower bound has not been set, a large negative number is returned.
- `XUI` output
  The upper bound. If an upper bound has not been set, a large positive number is returned.

9.1.11 SUBROUTINE GETFIX

Definition:
```
SUBROUTINE GETFIX ( IX )
INTEGER IX(*)
```

Purpose: Returns the fix statuses for all the minimization parameters.

Arguments:

- `IX` output

  The fix statuses with `IX(`i`)` corresponding to the fix status of parameter x_i. Possible values are:

  - `IX(`i`)` $= 0$

    Parameter x_i is fixed.
  - `IX(`i`)` $= 1$

    Parameter x_i is not fixed.

  Array `IX` should have at least N storage locations available.

9.1.12 SUBROUTINE GETFX1

Definition:
```
SUBROUTINE GETFX1 ( I, IXI )
INTEGER I, IXI
```

Purpose: Returns the fix status for one of the minimization parameters.

Arguments:

- `I` input

  Index of the minimization parameter whose fix status is requested.
- `IXI` output

  Fix status for parameter x_{I}. Possible values are:

  - `IXI` $= 0$

    Parameter x_{I} is fixed.
  - `IXI` $= 1$

    Parameter x_{I} is not fixed.

9.1.13 SUBROUTINE GETNAM

Definition:
```
SUBROUTINE GETNAM ( NAME )
CHARACTER*(*) NAME(*)
```

Purpose: Returns the symbolic names of all the minimization parameters. Symbolic names can be assigned using the `GODFATHER` command.

Arguments:

- `NAME` output

  `NAME(`i`)` stores the symbolic name of parameter x_i. Array `NAME` should have at least N storage locations available, each of them being at least 10 characters long. If no symbolic name has been set for parameter x_i, `NAME(`i`)` is left blank.

9.1.14 SUBROUTINE GETNM1

Definition:
```
SUBROUTINE GETNM1 ( I, NAMEI )
INTEGER I
CHARACTER*(*) NAMEI
```

Purpose: Returns the symbolic name of one of the minimization parameters.

Arguments:

- `I` input
  Index of the parameter whose name is requested.
- `NAMEI` output
  The symbolic name of parameter x_{I}. Variable `NAMEI` should be at least 10 characters long. If no symbolic name has been set for parameter x_{I}, `NAMEI` is left blank.

9.2 Panel related glue routines

These routines manipulate the values assigned to the keywords of the various panels. The `GETP` and `SETP` series of subroutines always operate on the panel of the current command, while the `GETPP` and `SETPP` series operate on any panel.

9.2.1 SUBROUTINE GETPI

Definition:
```
SUBROUTINE GETPI ( KEY, IVAL )
CHARACTER*(*) KEY
INTEGER IVAL
```

Purpose: Subroutine `GETPI` returns the integer value `IVAL` assigned to the keyword `KEY` in the current panel.

Arguments:

- `KEY` input
  The keyword to search for. It must exist in the panel of the current command and have an integer type, otherwise this routine will abort.
- `IVAL` output
  The integer value that corresponds to the keyword.

9.2.2 SUBROUTINE GETPR

Definition:
```
SUBROUTINE GETPR ( KEY, RVAL )
CHARACTER*(*) KEY
REAL RVAL
```

Purpose: Subroutine GETPI returns the floating point value RVAL assigned to the keyword KEY in the current panel.

Arguments:
- KEY input
  The keyword to search for. It must exist in the panel of the current command and have a real type, otherwise this routine will abort.
- RVAL output
  The real value that corresponds to the keyword.

9.2.3 SUBROUTINE GETPS

Definition:
```
SUBROUTINE GETPS ( KEY, SVAL, LES )
CHARACTER*(*) KEY, SVAL
INTEGER LES
```

Purpose: Subroutine GETPS returns the character string SVAL assigned to the keyword KEY in the current panel.

Arguments:
- KEY input
  The keyword to search for. It must exist in the panel of the current command and have a string or character type, otherwise this routine will abort.
- SVAL output
  The character that corresponds to the keyword. The length of SVAL should should be sufficient in order to accommodate all possible strings (as defined in the corresponding panel description file entry). If the declared length of SVAL is less then needed, the returned string will be truncated to fit; without any notification however.
- LES output
  The effective length of SVAL.

9.2.4 SUBROUTINE GETPPI

Definition:
```
SUBROUTINE GETPPI ( PANEL, KEY, IVAL )
CHARACTER*(*) PANEL, KEY
INTEGER IVAL
```

Purpose: Subroutine GETPPI returns the integer value IVAL assigned to the keyword KEY in the panel specified by PANEL.

Arguments:
- PANEL input
  The panel to be searched. It must be a valid panel, defined in the panel description file.

- `KEY` input
  The keyword to search for. It must exist in the panel and have an integer type, otherwise this routine will abort.
- `IVAL` output
  The integer value that corresponds to the keyword.

9.2.5 SUBROUTINE GETPPR

Definition:
```
SUBROUTINE GETPPR ( PANEL, KEY, RVAL )
CHARACTER*(*) PANEL, KEY
REAL RVAL
```

Purpose: Subroutine `GETPPR` returns the floating point value `RVAL` assigned to the keyword `KEY` in the panel specified by `PANEL`.

Arguments:

- `PANEL` input
  The panel to be searched. It must be a valid panel, defined in the panel description file.
- `KEY` input
  The keyword to search for. It must exist in the panel and have a real type, otherwise this routine will abort.
- `RVAL` output
  The real value that corresponds to the keyword.

9.2.6 SUBROUTINE GETPPS

Definition:
```
SUBROUTINE GETPPS ( PANEL, KEY, SVAL, LES )
CHARACTER*(*) PANEL, KEY, SVAL
INTEGER LES
```

Purpose: Subroutine `GETPPS` returns the character string `SVAL` assigned to the keyword `KEY` in the panel specified by `PANEL`.

Arguments:

- `PANEL` input
  The panel to be searched. It must be a valid panel, defined in the panel description file.
- `KEY` input
  The keyword to search for. It must exist in the panel and have a string or character type, otherwise this routine will abort.

- `SVAL` output
  The string or character value that corresponds to the keyword.
- `LES` output
  The effective length of `SVAL`.

9.2.7 SUBROUTINE SETPI

Definition:
```
SUBROUTINE SETPI ( KEY, IVAL, IERR )
CHARACTER*(*) KEY
INTEGER IVAL, IERR
```

Purpose: Subroutine `SETPI` assigns the integer number `IVAL` to the keyword `KEY` in the current panel.

Arguments:

- `KEY` input
  The keyword to be associated with the integer value. It must exist in the panel of the command currently being executed.
- `IVAL` input
  An integer value to be associated with the keyword.
- `IERR` output
  An error code indicating success or failure. Possible values are:
  - IERR = 0
    The routine completed successfully.
  - IERR = −1
    The panel description file indicates that the value of `IVAL` is not allowed for the given keyword.

9.2.8 SUBROUTINE SETPR

Definition:
```
SUBROUTINE SETPR ( KEY, RVAL, IERR )
CHARACTER*(*) KEY
REAL RVAL
INTEGER IERR
```

Purpose: Subroutine `SETPR` assigns the floating point number `RVAL` to the keyword `KEY` in the current panel.

Arguments:

- `KEY` input
  The keyword to be associated with the real value. It must exist in the panel of the command currently being executed.

- RVAL input
  A real value to be associated with the keyword.
- IERR output
  An error code indicating success or failure. Possible values are:
  - IERR = 0
    The routine completed successfully.
  - IERR = −1
    The panel description file indicates that the value of RVAL is not allowed for the given keyword.

9.2.9 SUBROUTINE SETPS

Definition:
```
SUBROUTINE SETPS ( KEY, SVAL, LES, IERR )
CHARACTER*(*) KEY, SVAL
INTEGER LES, IERR
```

Purpose: Subroutine SETPS assigns the character string SVAL to the keyword KEY in the current panel.

Arguments:

- KEY input
  The keyword to be associated with the string value. It must exist in the panel of the command currently being executed.
- SVAL input
  A string to be associated with the keyword.
- LES input
  Effective length of SVAL.
- IERR output
  An error code indicating success or failure. Possible values are:
  - IERR = 0
    The routine completed successfully.
  - IERR = −1
    The panel description file indicates that the value of SVAL is not allowed for the given keyword.

9.2.10 SUBROUTINE SETPPI

Definition:
```
SUBROUTINE SETPPI ( PANEL, KEY, IVAL, IERR )
CHARACTER*(*) PANEL, KEY
INTEGER IVAL, IERR
```

Purpose: Subroutine `SETPPI` assigns the integer number `IVAL` to the keyword `KEY` in the panel specified by `PANEL`.

Arguments:
- `PANEL` input
  Name of the panel that contains the specified keyword.
- `KEY` input
  The keyword to be associated with the integer value.
- `IVAL` input
  An integer value to be associated with the keyword.
- `IERR` output
  An error code indicating success or failure. Possible values are:
  - `IERR` $= 0$
    The routine completed successfully.
  - `IERR` $= -1$
    The panel description file indicates that the value of `IVAL` is not allowed for the given keyword.

9.2.11 SUBROUTINE SETPPR

Definition:
```
SUBROUTINE SETPPR ( PANEL, KEY, RVAL, IERR )
CHARACTER*(*) PANEL, KEY
REAL RVAL
INTEGER IERR
```

Purpose: Subroutine `SETPPR` assigns the floating point number `RVAL` to the keyword `KEY` in the panel specified by `PANEL`.

Arguments:
- `PANEL` input
  Name of the panel that contains the specified keyword.
- `KEY` input
  The keyword to be associated with the real value.
- `RVAL` input
  A real value to be associated with the keyword.
- `IERR` output
  An error code indicating success or failure. Possible values are:
  - `IERR` $= 0$
    The routine completed successfully.
  - `IERR` $= -1$
    The panel description file indicates that the value of `RVAL` is not allowed for the given keyword.

9.2.12 SUBROUTINE SETPPS

Definition:
```
SUBROUTINE SETPPS ( PANEL, KEY, SVAL, LES, IERR )
CHARACTER*(*) PANEL, KEY, SVAL
INTEGER LES, IERR
```

Purpose: Subroutine SETPPS assigns the character string SVAL to the keyword KEY in the panel specified by PANEL.

Arguments:

- PANEL input
  Name of the panel that contains the specified keyword.
- KEY input
  The keyword to be associated with the string value.
- SVAL input
  An integer value to be associated with the keyword.
- LES input
  Effective length of SVAL.
- IERR output
  An error code indicating success or failure. Possible values are:
  - IERR = 0
    The routine completed successfully.
  - IERR = −1
    The panel description file indicates that the value of SVAL is not allowed for the given keyword.

9.2.13 SUBROUTINE CHANGE

Definition:
```
SUBROUTINE CHANGE ( ICODE )
INTEGER ICODE
```

Purpose: This routine interacts with the user and handles panel I/O for all panel commands. It presents the panel, accepts any user changes to the panel parameters, and stores the new values in the appropriate data structures. Command line arguments are handled as well.

Arguments:

- ICODE output
  An error code indicating success or failure. Possible values are:
  - ICODE = 0
    The routine successfully processed all changes to the panel parameters.

- ICODE $= -1$
  An end–of–file condition was encountered while trying to read from the MERLIN input file.
- ICODE $=$ *any other value*
  Some other error has occurred, for example the given value for a keyword was out of the allowed range.

9.3 Utility glue routines

9.3.1 SUBROUTINE UPPER

Definition:
```
SUBROUTINE UPPER ( CHA, LL )
CHARACTER*(*) CHA
INTEGER LL
```

Purpose: Converts a string of characters to upper case. Assumes the ASCII character set.

Arguments:

- CHA input–output
  On input the characters to be converted. On output lower case characters are converted to their upper case equivalents.
- LL input
  Effective length of CHA.

9.3.2 SUBROUTINE I2STR

Definition:
```
SUBROUTINE I2STR ( N, STR, LES )
INTEGER N, LES
CHARACTER*(*) STR
```

Purpose: Converts an integer to its string representation.

Arguments:

- N input
  The integer to be converted.
- STR output
  Character variable containing the string representation of N. One must supply adequate storage.
- LES output
  Effective length of STR.

9.3.3 SUBROUTINE TOINT

Definition:
```
SUBROUTINE TOINT ( STR, LE, N, IERR )
CHARACTER*(*) STR
INTEGER LE, N, IERR
```

Purpose: Converts an integer from its string to its numeric representation. Leading or trailing blank or tab characters in the string are ignored.

Arguments:

- `STR` input
  The string to be converted.
- `LE` input
  The effective length of `STR`.
- `N` output
  The integer representation of `STR`.
- `IERR` output
  An error code, indicating whether the conversion was successful. Possible values are:
  - `IERR` $= 0$
    Conversion was successful.
  - `IERR` $= -2$
    Conversion failed. `STR` does not contain a proper integer number.

9.3.4 SUBROUTINE TOREAL

Definition:
```
SUBROUTINE TOREAL ( STR, LE, R, IERR )
CHARACTER*(*) STR
INTEGER LE
REAL R
```

Purpose: Converts a real number from its string to its numeric representation. Leading or trailing blank and tab characters are ignored. The number should be of the form:

[+ | -] *int* . *frac* *exp*

where *int* is the integer part (a sequence of digits), *frac* is the fractional part (another sequence of digits) and *exp* is the exponent. The integer part *int* or the fractional part *frac* may be omitted, but not both. The exponent *exp* should have the form:

E | e | D | d [+ | -] *int*

Arguments:

- `STR` input
  The string to be converted.

- `LE` input
  Effective length of `STR`.
- `R` output
  The converted real.
- `IERR` output
  An error code, indicating whether the conversion was successful. Possible values are:
  - `IERR` = 0
    Conversion was successful.
  - `IERR` = −2
    Conversion failed. `STR` does not contain a proper real number.

9.3.5 FUNCTION ISCOMP

Definition: `INTEGER FUNCTION ISCOMP ( STR1, STR2 )`
`CHARACTER*(*) STR1, STR2`

Purpose: Performs a case insensitive comparison of the two character strings `STR1` and `STR2`. Returns 0 if they don't match, non-zero if they match.

Arguments:
- `STR1` input
  First string to be compared.
- `STR2` input
  Second string to be compared.

9.3.6 FUNCTION LENGTH

Definition: `INTEGER FUNCTION LENGTH ( STR )`
`CHARACTER*(*) STR`

Purpose: Returns the effective length of a character variable. The effective length is defined as the position of the last non–blank, non–tab character in the string. Note that this is not the same as the declared length of a character variable.

Arguments:
- `STR` input
  A character variable of arbitrary length.

9.4 Miscellaneous glue routines

9.4.1 SUBROUTINE GETDIM

Definition: `SUBROUTINE GETDIM ( NOVAR, NOTERM )`
`INTEGER NOVAR, NOTERM`

Purpose: Returns the number of parameters (dimensionality) of the objective function, and the number of terms if the function is a sum of squared terms. These numbers are provided by the user at the time `SUBROUTINE MERLIN` is called.

Arguments:

- `NOVAR` output
  Number of parameters of the objective function.
- `NOTERM` output
  Number of terms if the function is a sum of squares. Otherwise `NOTERM` is set to 0.

9.4.2 SUBROUTINE ARGNO

Definition: `SUBROUTINE ARGNO ( NARGS )`
`INTEGER NARGS`

Purpose: Returns the number of arguments in the current MERLIN command.

Arguments:

- `NARGS` output
  The number of arguments in the current MERLIN command. This number does not include the command name itself.

9.4.3 SUBROUTINE GETARG

Definition: `SUBROUTINE GETARG ( NA, ARG, LENARG )`
`INTEGER NA, LENARG`
`CHARACTER*(*) ARG`

Purpose: Returns an argument from the MERLIN command line.

Arguments:

- `NA` input
  Index of the argument to be returned with $0 \leq$ `NA` $\leq$ `NARGS`. (`NARGS` can be obtained with a call to `SUBROUTINE ARGNO`). When `NA = 0` the command name itself is returned.

- `ARG` output
  The `NA`th argument from the MERLIN command line. It must have adequate length to accommodate all possible arguments.
- `LENARG` output
  Effective length of `ARG`.

9.4.4 SUBROUTINE GETACC

Definition:
```
SUBROUTINE GETACC ( ACC )
REAL ACC
```

Purpose: Returns the relative machine accuracy as estimated by MERLIN, or as set by the `MACHINE_DIGITS` directive in the configuration file.

Arguments:
- `ACC` output
  The relative machine accuracy. It is the largest positive number for which under the machine's finite precision the following conditions hold:

$$1 + \mathtt{ACC} = 1$$
$$1 - \mathtt{ACC} = 1$$

9.4.5 SUBROUTINE GETMC

Definition:
```
SUBROUTINE GETMC ( BIG, SMALL )
REAL BIG, SMALL
```

Purpose: Returns some machine constants. These should be set in the MERLIN configuration file, as they are system and installation dependent.

Arguments:
- `BIG` output
  The largest floating point number this machine can handle.
- `SMALL` output
  The smallest positive non–zero floating point number this machine can handle.

9.4.6 SUBROUTINE GETCNT

Definition:
```
SUBROUTINE GETCNT ( NF, NFP, NG, NGP, NH, NHP, NJ, NJP )
INTEGER NF, NFP, NG, NGP, NH, NHP, NJ, NJP
```

Purpose: Returns the total and partial MERLIN counters. All MERLIN counters are initialized to zero when MERLIN starts and are incremented when one of the user supplied subprograms is called. In addition, the partial MERLIN counters are reset to zero each time a `RESET` command is issued. There is no way to set the counters via a glue routine.

Arguments:

- `NF` output
  Total number of evaluations of the objective function since MERLIN started (calls to the user supplied `FUNCTION FUNMIN`).
- `NFP` output
  Number of evaluations of the objective function since the last `RESET` command was issued.
- `NG` output
  Total number of gradient evaluations (calls to the user supplied `SUBROUTINE GRANAL`).
- `NGP` output
  Number of gradient evaluations since the last `RESET` command was issued.
- `NH` output
  Total number of Hessian evaluations (calls to the user supplied `SUBROUTINE HANAL`).
- `NHP` output
  Number of Hessian evaluations since the last `RESET` command was issued.
- `NJ` output
  Total number of Jacobian evaluations (calls to the user supplied `SUBROUTINE JANAL`).
- `NJP` output
  Number of Jacobian evaluations since the last `RESET` command was issued.

9.4.7 SUBROUTINE GETFLA

Definition:

```
SUBROUTINE GETFLA ( I, F )
INTEGER I
REAL F
```

Purpose: Returns the value assigned to one of the MERLIN flags.

Arguments:

- `I` input
  Index of the flag whose value is requested.
- `F` output
  Value of the I$^{\mathrm{th}}$ flag.

9.4.8 SUBROUTINE SETFLA

Definition:

```
SUBROUTINE SETFLA ( I, F )
INTEGER I
```

```
REAL F
```

Purpose: Assigns a value to one of the MERLIN flags.

Arguments:
- `I` input
  Index of the flag.
- `F` input
  Value to be assigned to the `I`$^{\text{th}}$ flag.

9.4.9 SUBROUTINE GETCFL

Definition:
```
SUBROUTINE GETCFL ( I, CFL )
INTEGER I
CHARACTER*(*) CFL
```

Purpose: Returns the value of one of the MERLIN character flags.

Arguments:
- `I` input
  Index of the character flag whose value is requested.
- `CFL` output
  Value of the `I`$^{\text{th}}$ character flag. It should be at least 30 characters long.

9.4.10 SUBROUTINE SETCFL

Definition:
```
SUBROUTINE SETCFL ( I, CFL )
INTEGER I
CHARACTER*(*) CFL
```

Purpose: Assigns a value to one of the MERLIN character flags.

Arguments:
- `I` input
  Index of the flag.
- `CFL` input
  Value to be assigned to the `I`$^{\text{th}}$ character flag. Its effective length should be less than 30 characters otherwise it will be truncated.

9.4.11 SUBROUTINE SETCOD

Definition:
```
SUBROUTINE SETCOD ( MERR )
INTEGER MERR
```

Purpose: Sets the error code to be returned to the MERLIN operating system upon termination of a plug–in module.

Arguments:

- `MERR` input

  The error code to be returned to the MERLIN operating system, upon termination of the plug–in. Although in general a non–zero value indicates that something went wrong, MERLIN uses the following conventions:

  - `MERR` $= 0$

    No error occurred during execution of the plug–in.
  - `MERR` $= -1$

    An end–of–file condition was encountered while trying to read from the MERLIN input unit.
  - `MERR` $= -2$

    Some other error occurred during execution of the plug–in.

  By following the above conventions, one makes sure that an error condition will be treated appropriately by the MERLIN error handling routines, taking into account the current processing mode (`IAF` or `BATCH`). If a plug–in module terminates without setting the error code, a value of zero is assumed.

Example:

```
SUBROUTINE PLUG1
do something here
IF some error THEN
      MERR = -2
ELSE
      MERR = 0
CALL SETCOD(MERR)
END
```

9.4.12 FUNCTION ISMCL

Definition: `INTEGER FUNCTION ISMCL ( )`

Purpose: Returns 1 if MERLIN is running an MCL program; 0 otherwise.

9.4.13 SUBROUTINE GETIOU

Definition:

```
SUBROUTINE GETIOU ( NIN, NOUT )
INTEGER NIN, NOUT
```

Purpose: Returns the MERLIN input–output units. In order to maintain compatibility with the rest of the environment, write operations on the MERLIN output unit should take in account the printout levels as returned by `SUBROUTINE GETPRL`.

Arguments:
- `NIN` output
  The MERLIN input unit. Input should always be read from this unit.
- `NOUT` output
  The MERLIN output unit. Output should always be written to this unit.

Example:
```
LOGICAL STRO, WEAK
CALL GETPRL(STRO,WEAK)
CALL GETIOU(NIN,NOUT)
IF (STRO) WRITE (NOUT,*) 'An informative message'
IF (WEAK) WRITE (NOUT,*) 'An error message'
```

9.4.14 SUBROUTINE GETPRL

Definition:
```
SUBROUTINE GETPRL ( STRO, WEAK )
LOGICAL STRO, WEAK
```

Purpose: Returns a set of logical variables that control the display of MERLIN output. In order to maintain consistency with the rest of the environment, the same conventions should be used in a plug–in module.

Arguments:
- `STRO` output
  Logical variable that controls the display of normal output. Possible values are:
  - `STRO = .TRUE.`
    Normal output should be displayed.
  - `STRO = .FALSE.`
    Normal output should be inhibited.
- `WEAK` output
  Logical variable that controls the display of error output. Possible values are:
  - `WEAK = .TRUE.`
    Error output should be displayed.
  - `WEAK = .FALSE.`
    Error output should be inhibited.

Example: See `SUBROUTINE GETIOU`.

9.4.15 SUBROUTINE GETVAL

Definition: `SUBROUTINE GETVAL ( VALUE )`
`REAL VALUE`

Purpose: Returns the current value of the objective function. Note that you cannot request the value of the objective function if it has not been evaluated at least once. You can test this using the MERLIN call counters.

Arguments:
- `VALUE` output
  The current value of the objective function.

9.4.16 SUBROUTINE SETVAL

Definition: `SUBROUTINE SETVAL ( VALUE )`
`REAL VALUE`

Purpose: Sets the current value of the objective function.

Arguments:
- `VALUE` input
  The current value of the objective function. Note that this value *must* correspond to the current values of the minimization parameters (for example those set by `SUBROUTINE SETX`).

9.4.17 SUBROUTINE GETEVM

Definition: `SUBROUTINE GETEVM ( IEV )`
`INTEGER IEV`

Purpose: Returns the current function evaluation mode. The function evaluation mode can be changed with the `NOEVAL` and `EVALUATE` commands.

Arguments:
- `IEV` output
  The function evaluation mode. Possible values are:
  - `IEV` = 0
    Evaluation of the objective function has been disabled by a `NOEVAL` command.
  - `IEV` = 1
    Evaluation of the objective function is enabled.

9.4.18 SUBROUTINE GETFFO

Definition:
```
SUBROUTINE GETFFO ( IFF )
INTEGER IFF
```

Purpose: Returns the form of the objective function. The form of the objective function is chosen by initially supplying the appropriate subprogram (`FUNCTION FUNMIN` or `SUBROUTINE SUBSUM`) and then by choosing it by the `GENERAL` and `SOS` commands correspondingly.

Arguments:

- `IFF` output
  `IFF` indicates the form of the objective function. Possible values are:
  - `IFF` = 0
    Command `GENERAL` has been issued, indicating that the user has prepared the appropriate subprogram for a general function (`FUNCTION FUNMIN`).
  - `IFF` = 1
    Command `SOS` has been issued, indicating that the user has prepared the appropriate subprogram for a sum–of–squares function (`SUBROUTINE SUBSUM`).

9.4.19 SUBROUTINE GETTRG

Definition:
```
SUBROUTINE GETTRG ( TRG, ISTARG )
REAL TRG
INTEGER ISTARG
```

Purpose: Returns the current target value, as defined with a `TARGET` command. A common use of target values is as a termination criterion in a minimization routine. The target value can be set with the `TARGET` command and cleared with `NOTARGET`.

Arguments:

- `TRG` output
  The current target value. If there is no current target value (`ISTARG` = 0) the value returned depends on the current function mode:
  - `TRG` = *A large negative number*
    When MERLIN operates on a general function.
  - `TRG` = 0
    When MERLIN operates on sum–of–squares function.
- `ISTARG` output
  Indicates whether a target value has been set. Possible values are:
  - `ISTARG` = 0
    There is no target value currently in effect.

- ISTARG = 1
  A target value is in effect and is returned in variable TRG.

9.4.20 SUBROUTINE SETADE

Definition: SUBROUTINE SETADE (IAUTO)
INTEGER IAUTO

Purpose: Enables or disables the MERLIN automatic derivatives. A common use would be to enable the MERLIN automatic derivatives before calling a minimization routine that uses the gradient, and disable them immediately after. Note that while automatic derivatives are in effect, their options are taken from the MAD panel. Most of the MERLIN minimization commands have panel keywords that control the use of automatic derivatives during minimization.

Arguments:

- IAUTO output
  Specifies whether the MERLIN automatic derivatives should be enabled or disabled. Possible values are:
  - IAUTO = 0
    Automatic derivatives are disabled.
  - IAUTO = 1
    Automatic derivatives are enabled.

Example: CALL SETADE(1)
call some user minimization routine
CALL SETADE(0)

9.4.21 FUNCTION NUNIT

Definition: INTEGER FUNCTION NUNIT ( )

Purpose: Function NUNIT will search a range of unit numbers in order to locate an unused unit (a unit not assigned to an open file). Function NUNIT is a necessity since Fortran requires both, a unit number and a file name in an OPEN statement. Nevertheless function NUNIT and all of the MERLIN file handling routines rely on the Fortran INQUIRE statement and especially on the OPENED, EXIST, NUMBER and IOSTAT specifiers. The range of numbers to be searched can be set from the configuration file using the UNIT_RANGE directive. For efficiency, the search for the unit number starts from the number returned in the previous NUNIT call, not from the beginning of the available range.

Example:
```
NU = NUNIT()
IF (NU.EQ.-1) THEN
    display error message
ELSE
   OPEN (NU, FILE=some_file)
END IF
```

9.4.22 SUBROUTINE BACKUP

Definition:
```
SUBROUTINE BACKUP ( WHEN )
CHARACTER WHEN
```

Purpose: Adds a record to the backup file according to the modes specified in the backup panel.

Arguments:

- `WHEN` input
  Character variable, indicating the reason for the backup. If it matches the ones defined in the backup panel, then backup will proceed. MERLIN uses the following conventions:
  - `WHEN = 'M'` or `WHEN = 'm'`
    A backup record is added when a minimization command terminates.
  - `WHEN = 'X'` or `WHEN = 'x'`
    A backup record is added before the current point is changed by a `POINT`, `INIT` or `PICK` command.
  - `WHEN = 'D'` or `WHEN = 'd'`
    A backup record is added before one of the attributes (margin, fix status, or symbolic name) of the minimization parameters is changed.
  - `WHEN = 'L'` or `WHEN = 'l'`
    A backup record is added when a lower value is discovered by one of the minimization routines.

9.4.23 FUNCTION ISIAF

Definition: `INTEGER FUNCTION ISIAF ( )`

Purpose: Returns 1 if the `IAF` processing mode is in effect; 0 otherwise. Processing modes are switched using the `IAF` and `BATCH` commands.

9.4.24 FUNCTION ACSQ

Definition:
```
REAL FUNCTION ACSQ ( X, N )
INTEGER N
REAL X(N)
```

Purpose: All MERLIN minimization methods that operate on general functions (not on sum of squares) call this routine. It contains a counter and updates some common block parameters. Its value is set equal to the user supplied `FUNCTION FUNMIN` or `SUBROUTINE SUBSUM`, according to the functional form (`GENERAL` or `SOS`).

Arguments:

- `X` input
  The minimization parameters with `X(`i`)` corresponding to x_i.
- `N` input
  Dimensionality of the objective function.

9.4.25 SUBROUTINE LSQFCN

Definition:
```
SUBROUTINE LSQFCN ( M, N, X, F )
INTEGER M, N
REAL X(N), F(M)
```

Purpose: All MERLIN minimization methods that operate on functions that are sum of squares, call this routine. It contains a counter and updates some common block parameters. Its value is set equal to the user supplied `SUBROUTINE SUBSUM`.

Arguments:

- `M` input
  Number of terms in the sum-of-squares.
- `N` input
  Dimensionality of the objective function.
- `X` input
  The minimization parameters with `X(`i`)` corresponding to x_i.
- `F` output
  Partial terms of the sum of squares with `F(`i`)` corresponding to f_i.

Appendix A

MERLIN Quick Reference

MERLIN Quick Reference

Parameter related commands	*Description*	*Abbrev.*	*Page*
`POINT` *index value ...*	Assigns values to the parameters	`PO`	40
`LMARGIN` *index value ...*	Sets lower bounds on the parameters	`LM`	42
`RMARGIN` *index value ...*	Sets upper bounds on the parameters	`RM`	42
`LDEMARGIN` *index ...*	Removes the lower bounds from the parameters	`LD`	42
`RDEMARGIN` *index ...*	Removes the upper bounds from the parameters	`RD`	42
`FIX` *index ...*	Fixes some of the parameters to their current values	`FIX`	42
`LOOSE` *index ...*	Looses some of the parameters	`LOOSE`	43
`FIXALL`	Fixes all parameters	`FIXA`	43
`LOOSALL`	Looses all parameters	`LOOSA`	43
`GODFATHER` *index name ...*	Assigns symbolic names to the parameters	`GOD`	43
`NONAME` *index ...*	Removes the symbolic names from the parameters	`NON`	43
`INIT` *(panel command)*	Reads the parameter values or their attributes from a file	`INIT`	40
`PICK` *(panel command)*	Picks a record from a MERLIN file	`PI`	41
`TITLE` *Title for this session*	Sets a short title for the current MERLIN session	`TI`	44
`RESET`	Resets the partial call counters to zero	`RES`	45

Minimization and related commands	*Description*	*Abbrev.*	*Page*
`BFGS` *(panel command)*	Invokes the `BFGS` minimization algorithm	`BF`	49
`DFP` *(panel command)*	Invokes the `DFP` minimization algorithm	`DF`	51
`TRUST` *(panel command)*	Invokes the `TRUST` minimization algorithm	`TR`	53
`TOLMIN` *(panel command)*	Invokes the `TOLMIN` minimization algorithm	`TO`	51
`CONGRA` *(panel command)*	Invokes the `CONGRA` minimization algorithm	`CONG`	55
`ROLL` *(panel command)*	Invokes the `ROLL` minimization algorithm	`RO`	45
`SIMPLEX` *(panel command)*	Invokes the `SIMPLEX` minimization algorithm	`SI`	46
`LEVE` *(panel command)*	Invokes the `LEVE` minimization algorithm	`LE`	58
`AUTO` *(panel command)*	Invokes the `AUTO` minimization algorithm	`AU`	59
`ACCUM` *(panel command)*	Invokes the `ACCUM` minimization algorithm	`AC`	60
`TARGET` *target_value*	Sets a target value for the minimization methods	`TA`	61
`NOTARGET`	Clears the target value	`NOT`	62
`STEP` *index value ...*	Assigns values to the `ROLL` search steps	`STEP`	62

Merlin Quick Reference

`STEPDIS` [*index* ...]	Displays the `ROLL` search steps	`STEPD`	62
`ADJUST`	Constructs search steps for the `ROLL` method	`AD`	62
`STEPALL`	Constructs search steps for the `ROLL` method	`STEPA`	62
Informational commands	*Description*	*Abbrev.*	*Page*
`SHORTDIS` [*index* ...]	Displays the current values and attributes of the parameters	`SH`	44
`VALDIS`	Displays the current value	`V`	44
`TERMDIS` [*index* ...]	Displays the values of the M terms $f_i(\boldsymbol{x})$	`TE`	44
`MODEDIS`	Displays the current modes of operation	`MO`	71
`LIMITS`	Displays the build–in Merlin limits	`LIM`	71
Derivative related commands	*Description*	*Abbrev.*	*Page*
`FAST`	Forward differences are used to estimate the derivatives	`FA`	63
`QUAD`	Central differences are used to estimate the derivatives	`QUA`	63
`NUMER`	A higher order symmetric formula is used to estimate the derivatives	`NU`	63
`MIXED` *index mode* ...	Allows each gradient component to be estimated using a different mode	`MI`	64
`ANAL`	The user written `SUBROUTINE GRANAL` is used to calculate the derivatives	`AN`	63
`JANAL`	The user supplied `SUBROUTINE JANAL` is use to calculate the Jacobian	`JA`	64
`JNUMER`	The Jacobian is estimated numerically using forward differences	`JN`	64
`JCOMPARE`	Compare the numerically estimated Jacobian to the one supplied by the	`JC`	67
`GRADDIS` [*index* ...]	Displays the gradient components	`GRADD`	66
`GRADCHECK` $mode_1$ $mode_2$ [*index* ...]	Compares two different modes of calculating the gradient	`GRADC`	66
`GNORM`	Displays the gradient norms	`GN`	67
`HESSIAN` *(panel command)*	Operations related to the Hessian matrix	`HES`	64
`MAD` *(panel command)*	Configures the Merlin Automatic Derivatives	`MAD`	67
Mode commands	*Description*	*Abbrev.*	*Page*
`IAF`	Selects interactive operation	`IA`	68
`BATCH`	Selects batch operation	`BAT`	68
`FULLBACK`	Keeps all backup records	`FULLB`	68
`LASTBACK`	Keeps the most recent backup record	`LA`	68
`NOBACK`	Does not keep backup records	`NOB`	68
`BACKUP` *(panel command)*	Configures the Merlin backup mechanism	`BAC`	69

MERLIN Quick Reference

FULLPRINT	Allows full output to be displayed	FULLP	70
HALFPRINT	Allows only error messages to be displayed	HA	70
NOPINT	Suppresses all output	NOP	70
GENERAL	Instructs MERLIN to call the user supplied FUNCTION FUNMIN	GE	70
SOS	Instructs MERLIN to call the user supplied SUBROUTINE SUBSUM	SO	70
EVALUATE	Enables function evaluation	EV	71
NOEVAL	Disables function evaluation	NOE	71
File related commands	*Description*	*Abbrev.*	*Page*
DISCARD *file_name*	Overwrites a file with the current point and attributes	DI	75
DELETE *file_name ...*	Permanently removes a file from disk	DE	75
DUMP *(panel command)*	Writes the current point or any attribute in a file	DU	74
MEMO *(panel command)*	Stores the current point and attributes to a file	ME	73
INSPECT *(panel command)*	Inspects the contents of a file	INS	76
GOEOF *file_name ...*	Positions a file to the end–of–information	GOE	76
REWIND *file_name ...*	Rewinds a file	REW	76
Miscellaneous commands	*Description*	*Abbrev.*	*Page*
FLAG *index value ...*	Assigns values to the MERLIN flags	FLAG	62
CFLAG *index character_value ...*	Assigns values to the MERLIN character flags	CFLAG	63
FLAGDIS [*index ...*]	Displays the MERLIN flags	FLAGD	63
CFLAGDIS [*index ...*]	Displays the MERLIN character flags	CFLAGD	63
ALIAS *alias_name command*	Defines an alias	ALIAS	71
UNALIAS *alias_name*	Removes an alias definition	U	71
ALIASDIS	Displays the currently defined aliases	ALIASD	72
GRAPH *(panel command)*	Makes a rough terminal graph of the objective function	GRAP	77
PSGRAPH *(panel command)*	Makes a PostScript graph of the function or any X-Y data file	PSG	78
STOP [NOEPILOG]	Terminates execution	STO	72
RETURN [NOEPILOG]	Returns control to the program that called SUBROUTINE MERLIN	RET	72
QUIT *return_flag* [NOEPILOG]	Returns control to the program that called SUBROUTINE MERLIN	QUI	72
PANELON [*panel_command ...*]	Turns on panels	PANELON	81
PANELOFF [*panel_command ...*]	Turns off panels	PANELOF	81

MERLIN Quick Reference

`PSTATUS` [*panel_command ...*]	Displays the panel status	`PST`	81
`PDUMP` [*file_name panel_command ...*]	Writes panel parameters to a file	`PD`	81
`HIDEOUT` *file_name* [`APPEND`]	Redirects MERLIN output to a file	`HID`	82
`REVEAL`	Cancels the effect of `HIDEOUT`	`REV`	82
`HELP` *command*	Issues help texts for any MERLIN command	`HEL`	84
`LIST`	Lists all MERLIN commands	`LIS`	84
`MACRO`	Initiates a macro construction sequence	`MAC`	82
`CLEAR`	Terminates a macro construction sequence	`CL`	83
`RUNMCL` *object_code_file*	Initiates execution of a compiled MCL program	`RU`	83
`COVARIANCE` *(panel command)*	Calculates the covariance matrix	`COV`	83
`CONFIDENCE` *index ...*	Calculates confidence intervals	`CONF`	84
`HISTORY` *(panel command)*	Enables / disables the history mechanism	`HIS`	89
`ECHO` *any_message*	Displays any message	`EC`	86
`CONTROL` *(panel command)*	Sets some less used MERLIN options	`CONT`	87
`EPILOG` *epilog_command*	Defines an epilog	`EP`	87