MLwiN 2.0 Command Manual
Version 2.0.01
October 2003

Jon Rasbash
William Browne
Harvey Goldstein

Centre for Multilevel Modelling
Institute of Education
London, WC1H 0AL

Table of Contents

Commands - an introduction

1.1 Command: Introduction to MLwiN commands

Note: With release 2.0 nearly all of the MLn commands were incorporated into the graphical interface. Most standard analyses can be carried out without accessing the **command interface**. For those users who wish to write macros, or to carry out more advanced procedures the full range of commands is still available. *If you wish to make use of commands see getting started with commands .*

A list of the remaining commands that may sometimes be useful can be found by looking at commands for use in MLwiN 2.0

MLwiN and MLn

MLwiN is the successor to *MLn*, an established DOS based program. It performs multilevel analysis of data with any number of levels, together with associated data manipulation, tabulation, basic statistical functions, and graphing. *MLwiN* provides the same basic statistical functionality as *MLn* together with several new features including the ability to carry out Markov Chain Monte Carlo (MCMC) estimation and bootstrapping. The major innovation is the provision of a graphical user interface (GUI) which allows model definition and analysis without resort to *MLn* commands.
MLwiN has two components, a front end and a back end. The front end provides the GUI based interface to the user. The back end, is primarily the *MLn* DOS command based program with an interfacing module added, which allows it to receive requests for actions to be executed from the front end and to pass back the results of actions to the front end. There is also a command interface window, accessed from the data manipulation menu, which allows the user to issue commands as in *MLn*. These include the original MLn commands, even though some of these are now redundant, together with new commands which allow manipulation of aspects of the GUI, especially for users interested in writing macros. Many of these commands will, if issued in the **command interface**, update the relevant screens so that there results can be viewed there - many of *MLwiN* 's features are controlled via such commands.
The front end's requests are mediated by sending commands or groups of commands to the back end. For example, adding explanatory variables by clicking on objects in the equations window will result in the EXPL command being sent to the back end. Conversely if the user issues the EXPL command, in the **command interface** window or by obeying a macro containing the EXPL command the equations window will be properly updated.
The commands are grouped into sections according to their function, and appear in alphabetical order within each section which corresponds to a book in the help system. If you are new to multilevel modelling or have not used *MLn* you are advised to work through the introductory tutorial in the *MLwiN* user's guide.
*When searching for a particular command using the **index** tab in the help menu you should preface the command name by the word "command".*

Compatibility between *MLwiN* and MLn

Worksheets saved under *MLn* are recognised by *MLwiN* and can be retrieved. You may also save worksheets in MLn format.
Related topics are:
The MLwiN back end design
A summary of Macro commands
Command structures and definitions

Description of command parameters

1.2 Command The design of the MLwiN back end

In this section we describe the structure of the *MLn* back end.
The commands usually operate on data, which is held in a *worksheet*.
When *MLwiN* is first started the worksheet is empty and consists of a number of *columns* with labels C1, C2, ... and a number of *boxes* with labels B1, B2, ... These are the basic storage spaces of *MLwiN*. A column is designed to hold one or more *numbers,* each occupying one *cell* of the worksheet. Typically a column is used to hold the values of a variable, identifiers, predicted values, residuals, and codes for complex manipulation of data. A box holds one numeric value, for example a single statistic or a loop counter. Initially the columns are empty and the boxes each contain the system's UNKNOWN value. Arithmetic using this value will generate the UNKNOWN value. The elementary statistical operations of *MLwiN* will ignore it. It must not be present in any data which form part of a multilevel model.
It is often useful to think of columns in groups. They can also be linked formally into a *group* with one of the available labels G1, G2, ... Items in the same position in each of the columns of a group form a *row* and for some purposes may be interpreted as a *record*. Although such rows or records do not themselves have names or labels we use these terms informally to describe what is happening to the data.
MLwiN provides 1500 columns, 400 boxes, and 20 group labels by default. Also by default a model can specify up to 150 explanatory variables and up to 5 levels. The default size of a worksheet is 1,000,000 cells. All defaults except the number of boxes can be varied for a particular worksheet by the INITialise command or by accessing the **worksheet** on the **options** menu. Unless stated otherwise, the descriptions which follow assume that the default values are in force. If you issue a command which refers to a storage location or a level which is outside the limits in force for your worksheet, *MLwiN* will return the error message
`Wrong parameters`
Certain columns with numbers above 1000 are reserved. *MLwiN* uses these columns to store the latest estimates of the parameters of the current model, and their covariance matrices. The most important ones are as follows:

C1096	random parameter estimates
C1097	covariance matrix of random parameter estimates (lower triangle)
C1098	fixed parameter estimates
C1099	covariance matrix of fixed parameter estimates (lower triangle)

1.3 Command Macros - a summary

MLwiN has a macro language and macros are used to carry out certain kinds of analysis as described in the section(below) on macros. The standard macros supplied with *MLwiN* use particular columns, boxes, and groups. In addition, any other macros will use some of these: by convention only column numbers above 100 are used by macros. The columns, boxes and groups used by a macro should be displayed whenever a macro is run.
The Multilevel Models Project maintains up-to-date versions of all standard macros which are also distributed with the software. Both macros and documentation may be downloaded. from the MLwiN web site.
In release 2.0 and later of *MLwiN* the macros for fitting multilevel generalised linear models

are handled directly from the equations window . Further macros and instructions for their use are provided in a further document '*MLwiN macros for advanced multilevel modelling*'. These macros will fit models for time series in discrete or continuous time with continuous or discrete responses and event history (survival) models.
For details about macros see the topic *MLwiN* macro commands

1.4 Command structures and definitions

A command starts with a keyword describing its function, which may be followed by one or more parameters specifying the detail. A few commands, usually involving files, prompt the user for further information on one or more subsequent lines. Apart from these exceptions, and despite the length of some of the definitions in this manual, the complete command including all parameters must be typed on a single line.

Definitions

The command definitions contain

- bold capital letters **THUS** which must be typed as they appear (though not necessarily in capitals). These letters define the keyword, and four letters are always sufficient. Some keywords require only three letters.
- parameter descriptors in italic within angle brackets <*thus*> which denote constructs as explained below
- specific numerals in bold, for example **1**, or other text items such as **[**, which must be typed as they appear
- other explanatory text (including commas) which is optional

Parameter descriptors should be separated from one another and from other text by at least one space. If explanatory text is used it should not include numerals, labels or names which could be interpreted as parameters.

Alternative forms and optional parameters

Usually, alternative forms of a command are given separate definitions. Otherwise
| a vertical bar between two parameter descriptors indicates that a simple choice must be made between two forms of the parameter. Only one of the two should be entered.
{} braces enclose descriptors of parameters which for some purposes may be omitted altogether.
... an ellipsis following one or more parameter descriptors enclosed between braces indicates that further (sets of) parameters in the form within the braces may be entered on the same line.

For a description of how to use some basic commands see getting started with commands .

1.5 Command Parameter descriptors

The following table explains the parameter descriptors, etc.
The descriptors <*column*>, <*group*>, <string> and <*value*> are generic:
any descriptor including these words,
for example <*first column*>, <*explanatory variable group*>, etc.,

denotes a construct with the generic form described in the table
and with a meaning specific to the context of the command.

Descriptor Form and examples

<*box*> A box label of the form **B**<*value*> . For example B6 or BB3. A sequence of consecutively-labelled boxes may be entered by placing a hyphen between the first and the last, for example B13-B18 Note that there are no spaces around the hyphen.

<*C*> A column label of the form **C**<*value*> for example C10 or CB3

<*column*> A column label, or a name which has been previously attached to a column by the NAME command, or a construct of the form <*G*>**[**<*value*>**]** which specifies a column in a group, for example G7[B1]

<*filename*> The standard way of referring to a DOS file, for example c:\mln\test.ws

<*G*> A group label of the form **G**<*value*> for example G17 or GB10

<*group*> A list of one or more column labels, column names or group labels. A sequence of consecutively-labelled columns may be entered by placing a hyphen between the first and the last, for example 'var3'-'var7' or C4-C20 Note that there are no spaces around the hyphen. This range facility is not available for group labels. If you wish to refer to groups G3, G4, and G5 as a single group you must list them thus: G3 G4 G5

<*name*> A sequence of between 1 and 8 alphabetic, numeric, or underline characters enclosed in single quotes, for example 'age_1'

<*pathname*> The standard way of referring to a DOS path, for example c:\mydata. This will generally be of use only to writers of macros.

<*value*> A numeral or a box label

<*string*> MLwiN now has 20 string variables named S1..S20. At the moment there are only a few uses for these variables. In the future new commands will be added that will make greater use of these string variables. They are described in the section on new macro commands.

Note that in some cases we depart from this syntax where it gives rise to undue complexity and where a simpler clearly understandable syntax is available.

1.6 Commands for use in 2.0

Certain commands in version 2.0 have not been incorporated into the menu structure. A list of these, with brief descriptions is as follows. These will be of interest mainly to macro writers.

BOOT Selects a sample with replacement from a column
BSXE Used in cross classified models
CLEAr Clears all current model specifications
CTON Converts a categorical to numeric variable
DISCard Removes (numbered) rows from the stored data matrix
ERAS Erases specified columns
FCON Linear constraints on fixed parameters
FSDE Standard error type (robust, model based) for fixed parameters
LINK Assigns columns to a group
LOGA Appends output logging to existing file
LOGO Sets up a file for output logging

MARK Sets overwrite permission for a column
MIN, MAX Places minimum or maximum from a column into a box
MLRE Assigns numbering to a hierarchical structure
MOVE Moves high-numbered columns to fill unused column positions
MULS Forms random parameter design matrices for blocks using products
NTOC Changes variable from numeric to categorical
OFFSet Creates offset from residual SSP matrix
OLSE Fits an OLS regression model
PICK Picks a specified value from a column and copies into a box
RANKs Creates the ranks of the values in a column
RCON Linear constraints on random parameters
RSDE Standard error type (robust, model based) for random parameters
SEED Sets a seed for random number generation
SETD Sets up a user specified design matrix for random parameters
SETX Used in cross classified models
SIMU Simulates from a specified Normal model to an output column
SUBS Forms random parameter design matrices for blocks using sums
SURV Generates a vector for input to Cox survival models – see advanced macros (Yang et al., 1999)
TAKE Takes first value in specified set of blocks to new column
TIDY Organises worksheet to optimise space
WAIT Halts screen printing after each screenful.
WIPE Clears all data and model settings from worksheet
XOMIT Used in cross classified models
XSEArch Used in cross classified models

There is also a set of commands to manipulate the various matrices used by the IGLS algorithm.

1.7 Command - getting started

Getting started with commands

The remaining commands for use in version 2.0 which have no graphical interface equivalent fall into two groups; standard and advanced. The use of commands will be of interest mainly to users who wish to write macros .
If you wish to use commands and are new to them we suggest that you read the remainder of this topic and then click on the appropriate command in the table below to obtain details.
Commands are entered into *MLwiN* via a command interface window. Only the first 4 characters of a command are used. The lower, command entry box, is where user commands are entered and on pressing the 'return' key are executed and added to the list in the command storage box. You may reuse any command listed in this box simply by clicking on it, which highlights it and displays it in the command entry box. Pressing the return key then executes the command. You can also scroll back through the box to retrieve earlier commands.

If the user box is unchecked then, in addition to user specified commands, all commands issued by the MLwiN front end will also be displayed. (See introduction to MLwiN structure for further information on the structure of *MLwiN*) .
The output window is opened at the same time as the command input window. This is where all the commands and associated output in response to them will appear. The window will

contain all user generated commands and associated output since the start of the MLwiN session. A separate check box at the bottom of the output window specifies whether MLwiN front end generated commands and associated output will be displayed - but these will only be displayed following the checking of the box, not from the start of the session. This is useful, for example, if the user wishes to monitor error diagnostics issued by the MLn back end during the course of an iteration. Thus, if you suspect that a problem is occurring, say an iteration is proceeding very slowly, then stop after the iteration, open up the command interface output window, check this box and restart the iteration, leaving the window open. You will then see any error diagnostics being displayed.
All commands have the general form
KEYWord < first box, column, group, or number> < second box, column, group, or number> …… < final box, column, or group>
Thus, for example, if we wish to choose the tenth element in column 1 and place it into box 20 the command is
PICK 10 C1 B20
For a formal description of the structure of commands and definitions of groups and boxes see command parameter descriptors and command structure definitions . You may also wish to look at a description of how the *MLwiN* 'back end' which processes commands, is constructed.
The following two tables list the standard and advanced commands you may wish to use, together with brief descriptions and links to the formal descriptions of their syntax.

Standard Commands

LINK Assigns a group label to a set of columns. E.g. LINK C1 C2 G1, so that all subsequent references to G1 will refer to C1 and C2.
MARK Protects (or unprotects) data in a set of columns from inadvertently being overwritten.
TIDY Tidies the worksheet to allow more efficient use of space.
DISCard Eliminates a set of defined rows from a collection of columns.
PICK Picks a set of defined rows from a collection of columns.
MAX For each row of a set of columns, selects maximum to an output column
MIN For each row of a set of columns, selects minimum to an output column
SUMR Sums elements in a row of a set of columns and places result in output column.
TAKE Takes the first value in each unit, at a specified level, into an output column which will have length the total number of units at that level.
FCON Constrains a linear function of the fixed coefficients.
RCON Constrains a linear function of the random parameters.
FSDE Sets the type of standard errors to be used for fixed coefficients (e.g. model based or sandwich)
RSDE Sets the type of standard errors to be used for random parameters (e.g. model based or sandwich)
LGRId Evaluates the (Normal) likelihood over a grid of parameter values.
OFFSet Allows a pre-specified offset to be attached to the covariance matrix of the residuals, V.
OLSE Fits a simple multiple regression model for a specified response and explanatory variables.

Advanced Commands

Specifying cross-classified models: There is an example available.

SETX Sets up structure to run a cross-classified model
XOMIt Omits cells with less than a specified number of lowest level units.
XSEArch Searches structure for disjoint groupings.

BXSE Used to search efficiently for disjoint groupings. Not implemented in 1.1

Specifying multiple membership models: Click here for an introduction and example.

ADDM Adds indicator columns

WTCOl Assigns weights

Matrix operations Defining and manipulating matrices generally, and also matrices derived from a model.

Access to multilevel algorithm Provides access to the computational components of the IGLS algorithm

Other commands

MLREcode Recodes identifiers at a specified level to run consecutively

MULS Defines an explanatory variable for the random part of the model using lagged products of columns

SUBS Defines an explanatory variable for the random part of the model using lagged differences of columns

SURVival Converts a set of survival times, censored flags and input data into a form suitable for survival analysis with *MLwiN* . This has been incorporated into an advanced macro for survival (event history) analysis.

SETD Sets up a design vector specified by the user for a random parameter.

SEED Sets a seed for random number generator.

SIMU Simulates a response vector using current model parameters (assuming multivariate Normality).

BOOT Samples with replacement from a column.

Commands for arithmetic and data editing

2.1 Command ABSOlute values

ABSOlute values

ABSOlute values of <input group> to <output group> **ABSO**lute value of <*input value*> {to <*output box*>}

2.2 Command ACOS

ACOSine: Inverse cosine of <*input value*> (radians) {to <*output box*>}

2.3 Command ALOGit , antilogit

ALOGit , antilogit of values

ALOGit , antilogit of values in <input group> to <output group> ALOGit, antilogit of <*input value*> {to <*output box*>} ALOGit (*x*) = 1 / (1 + EXPO (-*x*))

2.4 Command ANGUlar transformation

ANGUlar transformation

ANGUlar transformation of values in <input group> results to <output group> ANGUlar transformation of <*input value*> {to <*output box*>} ANGU(*x*) = arcsin(*x*^0.5) in radians.

2.5 Command ANTIlogarithm, base ten,

ANTIlogarithm, base ten, of values in <input group> results to <output group>
ANTIlogarithm, base ten, of <*input value*> {to <*output box*>} ANTI(*x*) = 10^*x*

2.6 Command APPEnd

APPEnd to the ends of columns
APPEnd to the ends of columns in <input group> the columns in <appended group> to form <output group> For example, if C1 = (1 2 3), C2 = (4 5), C5 = (6 7), C7 = (8 9 10 11), then APPE C1 C2 C5 C7 C3 C4 produces C3 = (1 2 3 6 7), C4 = (4 5 8 9 10 11).

2.7 Command ASINe

ASINe: Inverse sine of <*input value*> (radians) {to <*output box*>}

2.8 Command ATAN

ATANgent: Inverse tangent of <*input value*> (radians) {to <*output box*>}

2.9 Command CALCulate

CALCulate {<box> = } <general expression >
CALCulate <*column*> | <*G*> = <*expression which may involve group labels, columns, boxes, and numerals*> For example CALC C1=0.56+SQRT(C2)-(C3^(2)*C4+LOGE(C5))/B2 In both formats <*expression*> typically includes operators, which may be logical, relational, or arithmetic. The operators and their rules of precedence are given in Appendix A. Parentheses () may be used, with their usual meaning. <*expression*> may also include the functions ABSOlute, ALOGit, ANGUlar, ANTIlogarithm, COSine, EXPOnential, LOGE, LOGIt, LOGTen, NEDeviate, ROUNd, SIGN, SINe, SQRT with arguments in parentheses (). In the first format <*expression*> may not include columns or groups. This is because a single value is computed. This value is printed on the screen and assigned to <*box*> if specified. In the second format <*expression*> may contain group labels, columns, boxes and numerals. If a group contains more than 1 column it is interpreted as a matrix, and its columns must be of equal length. An expression involving two operands and a binary operator, whether relational or arithmetic, is evaluated item by item. For example if C1 = (1 2 3), C2 = (1 4 5) and B2 = 3, CALC C3 = C1 + C2 produces C3 = (2 6 8) CALC C3 = (C1 != C2) * C2 produces (0 4 5) CALC C3 = (C1 <= 2) * (C2 + B2) produces C3 = (4 7 0) See the detailed description of the CALC command for further details, including matrix operations.

2.10 Command CATName

CATN mode N <input column> <category value, category name> <category value, category name>……
If Mode = 0 all category information is erased for <Input Column>
If Mode = 1 category values are assigned.

For example
CATN 1 C3 0 'boy' 1 'girl' will assign 0=boy and 1=girl.
If a string is identical to a column name then to avoid ambiguity place \ at the start of the string: the \ instructs *MLwiN* to treat the string as text rather than a column name.

2.11 Command CHANge

CHANge all items with value <value> in <input column> to the value <value>, results to <output column>
CHANge all items with values between <*value*> and <*value*> in <*input column*> to the value <*value*>, results to <*output column*> For example CHAN 2 C1 B1 C2 copies the contents of C1 to C2 and then changes all occurrences of 2 in C2 to the value in B1.

2.12 Command CHOOse

CHOOse
CHOOse only the items with value(s) <*value*> {to <*value*>} in <*input key column*> {carrying <*input group*>} and put into <*output key column*> {with carried values to <*output group*>} <*input group*> and <*output group*>, if specified, must have the same number of columns. <*input key column*> and <*output key column*> must be specified, and they may also belong to <*input group*> and <*output group*> respectively. For example, if C1 = (1 2 3 4 2 3 4 5), C2 = (1 2 3 4 5 6 7 8), C3 = (0 1 2 3 4 5 6 7), B7 = 3, then CHOO 1 B7 C1 C2-C3 C11 C12-C13 gives C11 = (1 2 3 2 3), C12 = (1 2 3 5 6), C13 = (0 1 2 4 5) If the values are such that nothing is chosen from the input key column then the output columns will all have zero length. See also OMIT.

2.13 Command CODE

CODE the numbers from one to <*value*>, each number repeated <*value*> times in a block, and repeat this set of blocks <*value*> times, results to <*output column*> For example CODE 3 4 2 C1 will produce the following codes in C1: 1 1 1 1 2 2 2 2 3 3 3 3 1 1 1 1 2 2 2 2 3 3 3 3

2.14 Command COSine

COSine of values (in radians) in <input group> to <output group>
COSine of <*input value*> (radians) {to <*output box*>}

2.15 Command COUNt

COUNt the items in <column> {, result to <box>}
If <*box*> is absent, display only.

2.16 Command CTON categorical to numeric

CTON C
This turns a categorical variable to a numerical variable and destroys the link between the value and category name assigned.

2.17 Command CUMUlative sum

CUMUlative sum of values in <input column> results to <output column>
The *m*th item in <*output column*> is set equal to the sum of the first *m* items of <*input column*>. For example if C1=(1 2 3 4) then CUMU C1 C2 gives C2=(1 3 6 10)

2.18 Command DISCard row(s)

DISCard row(s) number(s) <value> {to <value>} from <input group>, remainder to <output group>
DISCard rows whose numbers are in <*key column*> from <*input group*>, remainder to <*output group*> Note that it is the row positions within the columns of <*input group*> that are specified. Contrast with OMIT. See also PICK.

2.19 Command Divide

Divide
DIVI <value> by <value> result to <value>
For example DIVI C1 0.7 C2

2.20 Command EDIT

EDIT item number <value> in <column> to <value>
Note that the first <*value*> specifies the position of the item within <*column*> whose value is to be changed. Note also that this command operates directly on <*column*>.

2.21 Command EXPOnential

EXPOnential of values in <input group> to <output group>
EXPOnential of <*input value*> {to <*output box*>}

2.22 Command GADD across rows

GADD across rows
GADD C..C results to C Add across rows of input columns. Thus if C1 ={1,2}, C2={3,4} GADD C1 C2 C3 forms c3={4,6}

2.23 Command GENErate numbers

GENErate numbers
from one to <value> and store in <output column>
GENErate numbers from <*start value*> to <*end value*> {in steps of <*step value*>} and store in <*output column*> There are two forms of this command, the first with only one <*value*> specified, the second with two or three. In the first form the initial value and the step value are both assumed to be +1 and <*value*> should be a positive integer. In the second form none of the values need be positive, or integral. If <*step value*> is absent +1 is assumed. It should be possible to reach the <*end value*> from <*start value*> in a positive integral number of steps. For example GENE 5 C1 gives C1=(1 2 3 4 5), but GENE 1.5 7

2 C1 gives C1=(1.5 3.5 5.5)

2.24 Command GROUp

GROUp

GROUp the values in <*input column*> using the group upper boundaries stored in <*limit column*>, and place resulting group codes in <*code column*> The numbers in <*limit column*> must be in ascending order. Group codes will be integers from 1 to *n*+1, where *n* is the length of <*limit column*>. For example if C2 = (1.1 2 3.5) GROU C1 C2 C3 gives all values in C1 less than 1.1 a code of 1, all values greater than or equal to 1.1 and less than 2 a code of 2, all values greater than or equal to 2 and less than 3.5 a code of 3 and all values greater than or equal to 3.5 a code of 4. These codes will be stored in C3 in place of the corresponding values in C1.

2.25 Command integer divide

Integer divide

DIVide: integer divide. for example: 5 div 2 = 2

2.26 Command JOIN

JOIN

Join <value> | <group> {<value> | <group> ...} to form <output column> Take the values and the columns of the groups, in the order specified, and join them together (working down each column item by item and taking the columns in order within the groups) to form a single sequence, and place the result in <*output column*>. For example if C1=(1 2 3) and C2=(4 5) then JOIN 10 11 C1 C3 produces C3=(10 11 1 2 3) JOIN C1 10 11 C3 produces C3=(1 2 3 10 11) JOIN 10 C1 C2 11 C2 12 C3 produces C3=(10 1 2 3 4 5 11 4 5 12)

2.27 Command LISTwise delete

LISTwise delete records

LISTwise delete records which have value <value> in any of the columns of <input group>, results to <output group> For example LIST -9 C1 C2 C1 C2 will remove rows of values from C1 and C2 which contain -9 in either column. LISTwise delete records with values in <*key column*> in corresponding columns of <*input group*>, results to <*output group*> In this form, <*key column*> contains as many entries as there are columns in <*input group*>. Each entry is the missing-value code for the corresponding column in <*input group*>. This form is useful if different variables have different missing-value codes. For example if C2 = (99 5 3 -1) and C3 = (1 6 -1 5), and the missing-value code for C2 is 99 and for C3 is -1, set C1 to (99 -1). Then LIST C1 C2 C3 C4 C5 removes row 1 (missing value in C2) and row 3 (missing value in C3) producing C4 = (5 -1) C5 = (6 5).

2.28 Command LOGE logarithm base e

LOGE logarithm base e

LOGE logarithm base *e* of <*input group*> {to <*output group*>} **LOGE** logarithm base *e* of <*input value*> {to <*output box*>}

2.29 Command LOGIt

LOGIt of values

LOGIt **of values in <input group> to <output group>** LOGIt of <*input value*> {to <*output box*>} LOGI(*x*) = LOGE(*x*/(1-*x*))

2.30 Command LOGTen logarithm base 10

LOGTen logarithm base 10 of values in <input group> to <output group>

LOGTen logarithm base 10 of values in <input group> to <output group> LOGTen logarithm base 10 of <*input value*> {to <*output box*>}

2.31 Command MAXimum

MAXimum for each row of a set of columns of <input group> maximum to <output column>

Note: use only the first three characters MAX of the keyword.

2.32 Command MINimum

MINimum for each row of a set of columns of <input group> minimum to <output column>

2.33 Command MODulus

MODulus

The form of the command is:
Calc <Output column> = <Input Column> MOD <Divisor Column>
<Output column> contains the modulus remainder from the result of dividing <Input Column> by <Divisor Column>.
For example:
Calc b1 = 5 MOD 2
Produces b1 = 1

2.34 Command NTOC numeric to categorical

NTOC C

This turns a numeric variable into a categorical variable. Thus if C1 is called school and contains :

10
10
3
20
3

then NTOC C1 produces the following number->string mapping

10 -> SCHOOL_10

3->SCHOOL_3
20->SCHOOL_20

2.35 Command OMIT

OMIT
OMIT items with value(s) <*value*> {to <*value*>} from <*input key column*> {carrying <*input group*>}, remainder to <*output key column*> {with carried values to <*output group*>} <*input group*> and <*output group*>, if they are specified, must have the same number of columns. <*input key column*> and <*output key column*> must be specified, and they may also belong to <*input group*> and <*output group*> respectively. For example OMIT 1 C3 C1-C10 C13 C11-C20 will copy the records in C1-C10 across to C11-C20, omitting any records that contain a 1 in C3. OMIT 1 C3 C1-C2 C4-C10 C13 C11-C12 C14-C20 has the same effect. If the values are such that every value is chosen for omission from the input key column then the output columns will all have zero length

2.36 Command PICK item

PICK item number <value> from <input column> {and store value in <box>}
If <*box*> is omitted the item is displayed only.
PICK row(s) number(s) <*value*> {to <*value*>} from <*input group*> and store values in <*output group*>

2.37 Command PUT

PUT <value> items each with a value <value> into <column>
For example PUT 5 4.2 C5 gives C5 = (4.2 4.2 4.2 4.2 4.2)

2.38 Command RAISe values to power

RAISe values in <input column> to the power(s) specified by <index column> | <index value>, results to <output column>
RAISe <*input value*> to the power <*index value*> {and store in <*output box*>} For example if C1 = (2 3 4), C2 = (3 2 2), and B1 = 4, RAIS C1 B1 C3 gives C3 = (16 81 256) RAIS C1 C2 C3 gives C3 = (8 9 16) If you wish to raise to a power the values in a group containing more than one column you must issue a separate RAISe command for each column.

2.39 Command RANKs

RANKs of values in <input column> output to <output column>
The smallest value is given rank 1. Tied values are given average rank. For example if C1 = (2 1 1 3.2 3.1) RANK C1 C2 gives C2 = (3 1.5 1.5 5 4)

2.40 Command ROUNd

ROUNd values in <input group> to nearest integer, results to <output group>
ROUNd <*input value*> {to <*output box*>}

2.41 Command SET box

SET the value in <box> to <value>

For example SET B1 3 will set the contents of B1 to 3. SET B1 B15 will set B1 to the contents of B15. The command provides a quicker alternative to CALC B1 = 3 for example. Do not use SET if you wish to set a box equal to an expression: use CALCulate.

2.42 Command SIGN

SIGN of values in <input group> to <output group>

SIGN of <*input value*> {to <*output box*>} SIGN (*x*) = -1, 0, or 1 according as $x < 0$, $x = 0$, or $x > 0$.

2.43 Command SINe

SINe of values (in radians) in <input group> to <output group>

SINe of <*input value*> (radians) {to <*output box*>}

2.44 Command SORT

SORT

SORT on {<*value*>} key(s) in <*input key group*> {carrying <*input data group*>} results to <*output key group*> {with carried data to <*output data group*>} <*input key group*> must contain <*value*> columns (1 column if <*value*> is absent). <*output key group*> must contain the same number of columns as <*input key group*>. <*input data group*> and <*output data group*>, if specified, must have the same number of columns each. The first column of <*input key group*> is the major sort key. Further columns, if any, in <*input key group*> provide subsidiary sort keys in order of priority. Each column in <*input key group*> and <*input data group*> (if present) is sorted into the same order, based on this hierarchy of keys, and the results are placed in <*output key group*> and <*output data group*> respectively.

2.45 Command SPLIt

SPLIt

SPLIt the values in <*input column*> according to the codes assigned to them in <*code column*>, results to <*output group*> The lengths of <*input column*> and <*code column*> must be equal. The codes are the values in <*code column*> rounded to the nearest integer. <*output group*> must contain one column for each integer from the minimum code to the maximum. Each column of <*output group*> will contain values from <*input column*> corresponding to a single code in <*code column*>. If there are no such values the corresponding column will be empty. For example if C1=(1.1 2.3 3.1 1.9) and C2=(1 2 2 1) SPLI C1 C2 C3 C4 gives C3=(1.1 1.9), C4=(2.3 3.1)

2.46 Command SQRT

SQRT square root of values in <input group> to <output group>

SQRT square root of <*input value*> {to <*output box*>}

2.47 Command SUM

SUM the values in <column>, {result to <box>}

2.48 Command SUMRows

SUMRows in <group>, output to <column>
For each row in <*group*>, sum the elements and store the result in the corresponding position in <*column*>.

2.49 Command TANgent

Tangent: tangent of <*input value*> (radians) {to <*output box*>}

2.50 Command TRANspose

TRANspose the data in the <value> columns of <input group> to rows of <output group>
<*value*> must be the number of columns in <*input group*>. If all the columns of <*input group*> are of equal length *n*, <*output group*> must contain *n* columns. If the columns of <*input group*> are not of equal length the minimum length is used, and only that number of rows of data are transposed.

Elementary statistical commands

3.1 Command AVERage

AVERage values in <*data column*>, {using weights in <*weights column*>}, {count to <*box*> {mean to <*box*> {s.d. to <*box*> {s.e.m. to <*box*>}}}}
Store as many statistics, in the order given, as there are boxes specified. If no box is specified, display only.
AVERages and s.d. for the <*value*> columns in <*group*>, {means to <*column*> {s.d. to <*column*>}}
The number of columns in <*group*> must equal <*value*>. If no output column is specified, display only.

3.2 Command CHISquared

CHISquared for the two-way table containing the values in <group>
Compute Chi-squared statistic with continuity correction, and p-value, for overall test of independence.

3.3 Command CORMatrix

CORMatrix
of <value> variates in <group> {result to <output column>} Display correlation coefficients for each pair of variates in <*group*> as a lower triangle. Store the off-diagonal coefficients in stacked row order 21, 31, 32, 41, ..., in <*output column*> if specified.

3.4 Command CORRelate

CORRelate
<column-1> and <column-2> {with weights in <weights column>} {count to <*box*> {mean-1 to <*box*> {mean-2 to <*box*> {s.d.1 to <*box*> {s.d.2 to <*box*> {correlation coefficient to <*box*>}}}}}} For the values in <*column-1*> and <*column-2*> display count, means, s.d., correlation coefficient, and p-value for a test for a population correlation of zero. Store as many statistics, apart from the p-value, in the order given, as there are boxes specified on the command line.

3.5 Command CPRObability

CPRObability , chi squared tail probability
CPRObability tail probability for value <*value*> from the Chi-squared distribution with <*value*> degrees of freedom, {result to <*box*>} **CPRO**bability, tail probabilities for values in <*input column*> from the chi-squared distribution with <*value*> | <*column*> degrees of freedom, results to <*output column*>

3.6 Command DUMMy variables

DUMMy
DUMMy variables from values in <category column> results to < group> <*category column*> can contain any values: these values rounded to the nearest integer are considered as codes. The number of columns in < *group*> must be equal to, or 1 less than, the number of distinct integer codes that are produced by rounding the values in <*category column*>. If equal, a dummy variable is generated for each code; if 1 less, the lowest code is taken as the base category and dummy variables are generated for the remaining codes. For example if C1=(2 4 5) DUMM C1 C2-C4 gives C2=(1 0 0), C3=(0 1 0), C4=(0 0 1), DUMM C1 C2 C3 gives C2=(0 1 0), C3=(0 0 1).

3.7 Command FPRObability

FPRObability , F tail probability
FPRObability tail probability for value <*value*> from the F distribution with degrees of freedom <*value*> (numerator) and <*value*> (denominator), {result to <*box*>} **FPRO**bability, tail probabilities for values in <*input column*> from the F distribution with degrees of freedom <*value*> | <*column*> (numerator) and <*value*> | <*column*> (denominator), results to <*output column*>

3.8 Command GPRObability,

GPRObability, gamma distribution
GPRObability tail probability for value <*value*> from the Gamma distribution with shape parameter <*value*> and unit scale parameter {result to <*box*>} **GPRO**bability, tail probabilities for values in <*input column*> from the Gamma distribution with shape parameter(s) <*value*> | <*column*>, and unit scale parameter results to <*output column*>
The gamma distribution with shape parameter *a* is

$$f(x) = x^{a-1}e^{-x}/\Gamma(a)$$

3.9 Command HNORmal,

HNORmal,
Half-Normal scores of values in <input column> to <output column> A method of rescoring by assigning expected values in the upper half of the standard Normal distribution according to the ranks of the original scores. <*output column*> will contain the Normal Equivalent Deviates (NED) of $0.5(1+(i-0.5)/n)$ where *i* ranks the values in <*input column*> and *n* is the number of values. For example if C1 = (1 2 3 4 5 6 7 8 9 10) HNOR C1 C2 gives C2 = (0 ..... 1.96)

3.10 Command MOMEnts

MOMEnts
MOMEnts of values in <*input column*>, {k_1 (mean) to <*box*> {k_2 (variance) to <*box*> {k_3 to <*box*> {k_4 to <*box*>}}}} k_2, k_3, k_4 are unbiased estimates of 2nd, 3rd and 4th order cumulants. k_1 to k_4, together with skewness and kurtosis estimates (and their standard errors) are displayed. As many of k_1 to k_4 are stored, in order, as there are boxes specified.

3.11 Command NEDeviate,

NEDeviate, N(0,1) deviate
NEDeviate the N(0,1) distribution value corresponding to the cumulative probability <*value*>, {result to <*box*>} **NED**eviates, the N(0,1) distribution values corresponding to the cumulative probabilities in <*input group* > to <*output group* > For example if C1=(0.025 0.5) NED C1 C2 gives C2=(-1.96 0.0)

3.12 Command NPRObability,

NPRObability, Normal tail probability
NPRObability tail probability for value <*value*> from the standard Normal distribution, {result to <*box*>} **NPRO**bability, tail probabilities for values in <*input column*> from the standard Normal distribution, results to <*output column*>

3.13 Command NSCOres

NSCOres, Normal scores
NSCOres of the values in <input column> to <output column>
A method of rescoring by assigning expected values from the standard Normal distribution according to the ranks of the original scores. <*output column*> will contain the Normal Equivalent Deviates (NED) of $(i-0.5)/n$ where *i* ranks the values in <*input column*> and *n* is the number of values.
For example if C1 = (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)
NSCO C1 C2 gives C2 = (-1.96.....+1.96)

3.14 Command OREGress

OREGress as REGRess but through the origin.

3.15 Command REGRess:

REGRess:

REGRess <*response variate column*> {with weights in <*weights column*>}on <*value*> explanatory variables (excluding intercept) in <*explanatory variable group*> {putting predicted values in <*predicted value column*> {and coefficients in <*coefficients column*> (intercept last)}} <*value*> must match the number of columns in <*explanatory variable group*> and must not exceed 20. The analysis of variance table is displayed, together with a list of the regression coefficients and their standard errors.

3.16 Command TABStore:,

TABStore, tabulate and store

TABStore tabulate and store counts in <*output column*> using screen-output key <*value*> for a table with categories defined by the values in <*column*> **TABS**tore, tabulate and store counts in <*output column*> using screen-output key <*value*> for a table with categories defined by the values in <*column*> and <*column*> {for values in <*column*> equal to <*value*>} **TABS**tore, tabulate and store counts in <*output column*> using screen-output key <*value*> for a table with categories defined by the values in <*column*> and <*column*> {for values in <*column*> between <*value*> and <*value*>} **TABS**tore, tabulate, storing means in <*output column*>, the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> **TABS**tore, tabulate, storing means in <*output column*>, the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> and <*column*> {for values in <*column*> equal to <*value*>} **TABS**tore, tabulate, storing means in <*output column*>, the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> and <*column*> {for values in <*column*> between <*value*> and <*value*>} The parameters for TABStore are as for TABUlate, with the exception of the initial <*output column*>. The meaning of the other parameters, and the resulting screen display, are as for TABUlate. The counts, or (in the second main form) means, are stored in <*output column*>. For a two-way table, the first row of counts (or means) displayed is stored first, followed by the second, and so on.

3.17 Command TABUlate

TABUlate

TABUlate using output key <value> for a table with categories defined by the values in <column> **TABU**late using output key <*value*> for a table with categories defined by the values in <*column*> and <*column*> {for values in <*column*> equal to <*value*>} **TABU**late using output key <*value*> for a table with categories defined by the values in <*column*> and <*column*> {for values in <*column*> between <*value*> and <*value*>} **TABU**late the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> **TABU**late the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> and <*column*> {for values in <*column*> equal to <*value*>}.

TABUlate the means and s.d.s of values in <*variate column*> for each cell in the table defined by categories in <*column*> and <*column*> {for values in <*column*> between <*value*> and <*value*>} There are two main forms of this command, with three main variants of each. The absence of an initial parameter <*value*> implies the second main form. The categories are the values in the table-defining column(s) rounded to the nearest integer. Counts for each category are displayed in columns. For a 2-way table the rows of the display are defined by the

categories in the second defining column. The key <*value*>, if present, specifies information to be displayed in addition to the counts, as follows: 0: display no additional information 2: display percentages of column totals 1: display percentages of row totals 3: display percentages of the grand total These may be combined by placing together. For example key 12 displays percentages of rows and columns. For a 1-way table keys 1, 2, and 3 are equivalent. For a 2-way table key 4 displays Pearson chi contributions for each cell. Display can be restricted to entries with a specified value, or within a specified range, of a further variable, but only in the case of a 2-way table

3.18 Command TPRObability

TPRObability,

TPRO tail probability for <value> from t-distribution with <value> d.f., {result to <box>}
TPRObability, tail probabilities for values in <*input column*> from the t-distribution with <*value*> | <*column*> degrees of freedom, results to <*output column*>

3.19 Command ZSCOres

ZSCOres,

ZSCO Standardise the values in <input group> to have mean 0 and s.d. 1, results to <output group> ZSCO(x)=(x-mean(x))/s.d.(x)

Calculate command

4.1 Command CALC command examples

The CALCulate command treats groups as matrices. The operands we consider here are columns or single values (which may be expressed as numerals or boxes).
If all the operands are single values, that is either boxes or numerals, the evaluation of <*expression*> proceeds as in straightforward arithmetic and produces a single value as a result. This result may be assigned to a box, or it may simply be printed (see the first format of the CALC command). The result may not be assigned to a column. If it is desired to set up a column with a single value resulting from a calculation, first assign the value to a box and then use the JOIN command, for example:

```
eras c10
calc b4 = (b3 >= 1)*abso(b5) + (b3 < 1)*(-abso(b5))
join c10 b4 c10
```

will place in C10 the absolute value of B5 if B3 is greater than or equal to 1, and the negative absolute value of B5 otherwise. Note that two operators should not be coded together: hence the use of parentheses around the unary negative operator. It is not necessary to separate function names from other operators.
If one of the operands is a column then all other columns in <*expression*> must be of the same length, which also determines the length of the output column. (There are exceptions to this rule if the columns are interpreted as matrices .) A simple example is:

```
inpu c1 c2
1 3
4 2
6 8
9 7
3 18
calc c3=c1>c2 & c1<=6
```

```
prin c1 c2 c3
        C1           C2           C3
 N =      5            5            5
   1  1.0000       3.0000       0.0000
   2  4.0000       2.0000       1.0000
   3  6.0000       8.0000       0.0000
   4  9.0000       7.0000       0.0000
   5  3.0000       18.000       0.0000
```

The first relation compares each element of C1 in turn with the corresponding element of C2 and generates a set of 5 results (1 for *true*, 0 for *false*). The second relation compares each element of C1 in turn with the single value 6 and generates a second set of 5 results. These are then logically combined, item by item, with the corresponding result from the first relation.

Now:

```
calc c3=c1>c2 | c2==6*c1
prin c1-c3
        C1           C2           C3
 N =      5            5            5
   1  1.0000       3.0000       0.0000
   2  4.0000       2.0000       1.0000
   3  6.0000       8.0000       0.0000
   4  9.0000       7.0000       1.0000
   5  3.0000       18.000       1.0000
```

Here the first operation to be performed is 6*c1. This produces a working set of 5 values, each value 6 times the corresponding element in C1. The relational expressions are then evaluated, element by element, and finally combined.

Now

```
gene 5 c4
calc b1=4

 4.0000

calc c4 = c4 + 2 * c4 * (c2>=b1 & c1<b1)
prin c1 c2 c4
        C1           C2           C4
 N =      5            5            5
   1  1.0000       3.0000       1.0000
   2  4.0000       2.0000       2.0000
   3  6.0000       8.0000       3.0000
   4  9.0000       7.0000       4.0000
```

4.2 command EQMI for missing data

Used with calculate command
EQMI <N1>
N1=1 (default) for == or != operators if either operand is missing the result is missing
N1=0 for == or != operators if either operand is missing, then the missing value code is compared with the value in the operand; if true then result = 1, if not result =0

4.3 Command The CALCulate command

The CALCulate command

The format of the CALCulate command is one of: **CALC**ulate {<*box*> = } <*expression*>
CALCulate <*group label*> | <*column*> = <*expression*>
<*expression*> consists of one or more *operands* which may be operated on by one or more

operators expressing a calculation. Each operand must have the form <*value*>, <*column*>, or <*G*> .The result of the calculation must match the type specified on the left of the = sign or, if no = sign is present, the calculation must evaluate to a number.
The values of the operands in <*expression*> are not changed during the course of the calculation. Assignment to the group, column or box, if any, named on the left of the = sign, takes place only at the end of the calculation.
Operators, if present, can be arithmetic, relational, logical, or matrix. The arithmetic operators are:

+	add
-	subtract
*	multiply
/	divide
^	raise to the power
-	(unary) the negative of

The functions ABSOlute , ACOS, ASINe, ALOGit , ANGUlar , ANTIlogarithm , ATAN, COSine , EXPOnential , LOGE , LOGIt , LOGTen MODulus>generic2, MODulus , NEDeviate , ROUNd , SIGN , SINe , SQRT , TANgent also are available. These, like the unary negative above, operate on the operand immediately to their right and produce results as described in the sections on arithmetic and data-editing commands and on elementary statistical operations. The other arithmetic operators require two operands.
The relational operators are:

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
!=	is not equal to
==	is equal to

The logical operators are:

&	and
|	or
!	not

Each of these operators requires two operands and produces a result with the value 1 if the relation is *true*, 0 if *false*.
Finally, there are the following matrix operators:

*.	matrix multiply
~	transpose of
diag	diagonal of
det	determinant of
inv	inverse of
sym	symmetric matrix of
hsym	half-symm. matrix of

See help topics on these for details.
The order of precedence of the operators is:

level	operators
1 (low)	&, |
2	<, >, >=, <=, !=, ==
3	+, -
4	*, /

5 *.
6 unary -, ~, !
7 (high) named functions NED, INV etc.

Parentheses () may be used in the usual way to prioritise one or more operations. The arguments of named functions should always be placed in parentheses, as should the unary operators (level 6 above) with their operands.
For the == and != commands mising data in either operand propagates to result. This can be changed; see EQMI command

Commands for input and output of data

5.1 Command INCO1

INCOl C "filename" read numeric data from file into single column C, regardless of record structure in file

5.2 Command DELImit input or output

DELImit input or output fields according to <value>
Use in preparation for the commands FDOUtput or DINPut . <*value*> is the ASCII code for the required delimiter: commonly used ones are 9 (tab), 32 (space), 44 (comma). To remove a delimiter, type DELI 0 The DELImit command has no effect on the FDINput or DWRIte commands.

5.3 Command DINPut data

DINPut data to <group> from a file
Respond <*filename*> to prompt. The file <*filename*> must contain lines of numeric data, each pair of numbers separated by the character specified in the most recent DELImit command, if any, or (if none) by one or more spaces. The numbers will be placed one by one into successive columns of <*group*>, forming a row or part of a row. Any pre-existing contents of <*group*> will be over-written. If the last column of <*group*> is not reached during input of a line of numbers the row will continue with the numbers on the next line of <*filename*>. Once the last column of <*group*> has been reached any further numbers to be input must begin on a new line of <*filename*> and will begin the next row of <*group*>. A line of <*filename*> must not exceed 500 characters in length.

5.4 Command DWRIte data

DWRIte data in <group> to a file
Respond <*filename*> to prompt. The data in <*group*> are output row by row to the file <*filename*> in ASCII code. Each number is expressed to 5 significant figures and occupies 15 characters of the output line, at least 5 of which are leading spaces. At most 5 numbers are output to any line. If a row of <*group*> contains more than 5 numbers, they will be output 5

at a time until the row is finished. A new row of <*group*> always starts a new line of output.

5.5 Command FDINput formatted data

FDINput formatted data to <group> from a file

Respond <*filename*> to first prompt. Respond <*format*>, in parentheses, to second prompt. Each line of the file <*filename*> is interpreted as a record of data, or part of a record of data, each field of which is coded in a fixed column or set of adjacent columns in the line. <*format*> is a sequence of numbers separated by commas indicating how each record is to be read. It must be typed, together with its parentheses, on a single line. An oblique stroke / in place of a comma in <*format*> indicates that the record continues on the next line of data in <*filename*>. A positive number *m* indicates that the next *m* columns of the record are to be read as one numeric field and placed in the next column of <*group*>. A negative number -*m* indicates that the next *m* columns of the record are to be skipped. The number of positive numbers in <*format*> must match the number of worksheet columns in <*group*>. This command will cause any existing contents of <*group*> to be over-written. For example to read two variables held in columns 2-4 and 15-16 of each line of the file read.dat held in the current directory type: FDIN C1 C2 READ.DAT (-1,3,-10,2) If a record of data contains many fields, and you wish to read them all into the worksheet, it may be necessary to issue more than one FDINput command, using the associated <*format*> to skip over data previously read. A line of <*filename*> must not exceed 500 characters in length.

5.6 Command FDOUtput data

FDOUtput data in <group> to a file in specified format

Respond <*filename*> to first prompt. Respond <*format*>, in parentheses, to second prompt. Data in <*group*> are output in rows, each row occupying one or more lines of the file <*filename*>. <*format*> is a sequence of numbers separated by commas indicating how the data are to be spaced on each line of output. <*format*>, together with its parentheses, must be typed on a single line. A positive number *m* in <*format*> indicates that the next data item is to occupy the next *m* columns of the line. A negative number -*m* indicates that the next *m* columns of the line are to be skipped. If a delimiter has been specified by the DELImit command, this will be placed between each pair of output numbers on a line and occupy an extra column. An oblique stroke / in place of a comma terminates the current output line: data from further items in the current row of <*group*> will be output to the next line of <*filename*>. The number of positive numbers in <*format*> must match the number of columns in <*group*>. For example to write two variables held in C1 and C2 so that their values occupy columns 2-4 and 5-6 of each line of the file c:\testdata\write.dat type: FDOU C1 C2 C:\TESTDATA\WRITE.DAT (-1,3,2)

5.7 Command FINIsh

FINIsh

Terminates the input of numbers via the ASSIgn command.

5.8 Command INPUt numbers into <group>

INPUt numbers into <group>

This command is no longer available. Data input is via the FILE menu. In macros use the ASSIgn command instead.

5.9 Command PRINt on screen

PRINt on screen the contents of <box>|<group> {<box>|<group> ...}
A sequence of boxes can be indicated by, for example, B1-B10. The contents of all boxes specified are displayed on screen, followed by the contents, row by row, of each group. Numbers are displayed with 5 significant figures, at most 5 to a line. If the groups together contain more than 5 columns these will displayed 5 at a time. If waiting is on (see WAIT command), display will pause when the screen is full. Press RETURN to view the next screen, or Q to terminate the display.

5.10 Command VIEW text file

VIEW text file
Respond <*filename*> to prompt. Display the contents of <*filename*>, which will be interpreted by *MLwiN* as ASCII data.

Character plotting commands

6.1 Command BOXPlots

BOXPlots of values in <group>
Display standard boxplots for the variates in <*group*>, showing median, quartiles and maximum and minimum values, with a common scale determined by the overall maximum and minimum values in <*group*>.

6.2 Command HISTogram

HISTogram of values in <input column> {with interval width <value>} {frequencies to <output column>}
If interval width is not specified it is chosen automatically. The number of frequencies placed in <*output column*>, if specified, will equal the number of intervals. The lower limit of each interval is displayed. Intervals run from this limit up to but not including the next limit (if any). For example, if C1 = (1 1 1 2 2 3) then HIST C1 1 C2 gives C2 = (3 2 1)

6.3 Command LPLOt

LPLOt values in <Y-column> against values in <X-column> using codes in <code column> to assign a letter to each point
<*Y-column*>, <*X-column*> and <*code column*> must all be the same length. For example, if there are 5 points to be plotted, with *X*-coordinates in C1 and *Y*-coordinates in C2, and C3=(1 1 1 2 2), then LPLOt C2 C1 C3 will plot the first three points using the letter A and the last two using the letter B.

6.4 Command MPLOt

MPLOt sets of values in <Y-group> against values in <X-column>
Display multiple plots, one for each variate in <*Y-group*>, against the same *X*, on a single chart. Different alphabetic characters are used for each plot and the number of variates in <*Y-group*> must not exceed 26.

6.5 Command PLOT

PLOT

PLOT values in <*Y-column*> {with *Y*-axis range <*value*> to <*value*>} against values in <*X-column*> {with *X*-axis range <*value*> to <*value*>} Display a basic scatterplot.

6.6 Command STEM and leaf

STEM and leaf

STEM and leaf plot of values in <*input column*> {with interval width <*value*>} {frequencies to <*frequency column*> {and mid-points to <*mid-point column*>}} If interval width is not specified it is chosen automatically. The lower limit of each interval is displayed. Intervals run from this limit up to but not including the next limit (if any). The leaf unit is the power of 10 next lower than the interval width. For each interval the lower limit is displayed, together with the number of items in <*input column*> that fall within the interval. These form the stem. For each item in a given interval a digit is displayed as a leaf on the right of the stem at that level. The leaf is the offset of the item (in leaf units) from the lower limit of the interval. For example, if C1 = (1 1 1 2 2 3) then STEM C1 1 C2 C3 gives C2 = (3 2 1), C3 = (1.5 2.5 3.5) and displays a stem with lower limits 1, 2, and 3, and frequencies 3, 2, and 1. The leaves are all zeroes. If C2 = (-2.7 -2.7 -1.7 -0.7 -0.7 -0.7 0.3 1.3 1.3 1.3 2.3) STEM C2 automatically chooses an interval width of 1, and therefore a leaf unit of 0.1. Lower limits are -3, -2, -1, 0, 1, 2, with frequencies 2, 1, 3, 1, 3, 1, respectively. The leaves are all 3s.

Commands for model estimation and control

7.1 Command BATCh {<value>}

BATCh {<value>}

If <*value*> = 0, turn batch mode off. If <*value*> = 1, turn batch mode on. If <*value*> is absent, reverse the current mode.

7.2 Command CLEAr all model specifications

CLEAr all model specifications

The model specifications and the residual settings are cleared.

7.3 Command CLRDesign

CLRDesign

CLRDesign remove from the random part at level <*value*> the design vector corresponding to <*column*> This command undoes the effect of a previous SETDesign command. See SETDesign. After using SETDesign or CLRDesign it is recommended that you continue estimation by typing STARt, not NEXT.

7.4 Command CLRElements

CLREelements

CLRElements remove from the random part at level <*value*> the covariance matrix element(s) defined by the pair(s) of columns <*first column*> <*second column*> {<*first column*> <*second*

column> ...} This command is useful when it is required to estimate some, but not all, of the variances and covariances for a set of coefficients at a particular level. Note that each covariance element to be removed from the model, including that for a variance, requires two columns to be specified in order to define it. For example, CLRE 1 C1 C2 removes from the model the covariance at level 1 between the coefficients of the variables in these columns. The variances of both coefficients, if they are currently in the model, will remain.

7.5 Command CLRVariances

CLRVariances

CLRVariances and covariances at level <*value*> {for explanatory variables in <*group*>} Remove the variables in <*group*> from the random part of the model at level <*value*>. If <*group*> is not specified, remove all variables from the random part at level <*value*>. For example, if the level-2 random part of the current model contains C1, C2, and C3, and the full 3-by-3 covariance matrix for the coefficients is currently estimated, then CLRV 2 C2 removes from the estimation the variance of the coefficient of C2 at level 2 and its covariances at level 2 with the coefficients of C1 and C3. Thus the random part at level 2 now contains only C1 and C3.

7.6 Command Exclude

EXCLude mode N <input column>

If Mode = 1 level 1 units are excluded from the analysis where corresponding elements in <input column> are non-zero.
If Mode = 0 turn exclusion off.
If you specify through the graphs window that particular units at any level are to be excluded from the model then the corresponding elements in <input column> are set to have non-zero values with other elements unchanged. If you specify some units to be excluded and no exclusion column has been specified *MLwiN* chooses the last free column on the worksheet. The exclusion column can be set on the screen that appears when pressing the options button in the hierarchy viewer.
This command is useful if exclusion is to be based upon complex criteria, using the Calculate facility.

7.7 Command EXPLanatory

EXPLanatory

EXPLanatory variables are in <*explanatory variable group*> All explanatory variables in either the fixed or the random part of a model must first be declared as explanatory variables by the EXPL command. In the above form EXPL is a toggle command. Initially, when the model is empty, all variables in <*explanatory variable group*> will be declared as explanatory variables in the fixed part of the model (see also FPAR). If EXPL is used again in the above form, only those variables in <*explanatory variable group*> which are not currently declared will be declared: those that are already declared will be undeclared and removed from both the fixed and the random parts. **EXPL** mode <*value*> <*explanatory variable group*> If <*value*> = 1, ensure all variables in <*explanatory variable group*> are declared. If <*value*> = 0, ensure all variables in <*explanatory variable group*> are undeclared and removed from the fixed and the random parts. **EXPL** (with no parameters) Undeclare all explanatory variables and remove them from the fixed and random parts. You will be asked to confirm that you

wish to do this.

7.8 Command FCONstraints

FCONstraints

FCONstraints put constraints on the fixed parameters as specified in <*constraints column*> Suppose there are *p* fixed parameters and that *r* linear constraints on them are required. These are first expressed in the form $Ax=b$ where x is the p-by-1 vector of the parameters, A is an r-by-p matrix of multipliers of the parameters, and b is the r-by-1 constrained result of the multiplication. Thus the first row of A and the first element of b define the first linear relationship; and so on. <*constraints column*> contains the elements of $[A,b]$ stored row by row. Thus the first $p+1$ elements of <*constraints column*> are the first set of multipliers followed by the constrained value of the first linear combination; and so on. <*constraints column*> contains $r*(p+1)$ values. For example, if there are four fixed parameters and we wish to constrain the first three of them to be equal we may proceed as follows. $r=2$, $p=4$; let the first row of A be (1 -1 0 0), and the second row be (0 1 -1 0); b=(0 0)'. Thus we set <*constraints column*> to (1 -1 0 0 0 0 1 -1 0 0). Note that the 4th parameter, although unconstrained, still occupies spaces in the <*constraints column*>. **FCON** (with no parameters) Remove any active set of constraints from the fixed part of the model. An alternative form useful in macros is: **FCON** <*box*> Output the number of the current constraints column to <*box*>. If no constraints are specified, attach the last unused column on the worksheet to the model as a constraints column and return the column number <*box*>.

7.9 Command FIXEd

FIXEd

Display the current estimates of the fixed parameters

7.10 Command FPARt, amend the fixed part

FPARt, amend the fixed part of the model as specified by <group>

In this form FPAR is a toggle command. Any variables in <*group*> which are currently in the fixed part will be removed. Any which are not (i.e., have been previously removed) will be added, provided they are currently declared as explanatory variables (see EXPLanatory command).

FPAR mode <value> <group> If <*value*> = 1, ensure all variables in <*group*> which are currently declared as explanatory variables are included in the fixed part of the model. If <*value*> = 0, ensure all variables in <*group*> are removed from the fixed part (such variables remain declared as explanatory variables and may or may not be in the random part). **FPAR** (with no parameters) Remove all variables from the fixed part of the model.

7.11 Command FSDErrors type <value>

FSDErrors type <value>

Set the type of standard error computation for fixed parameters. If <*value*> = 0, use standard, uncorrected, IGLS or RIGLS computation. If <*value*> = 2, compute robust or 'sandwich' standard errors based on raw residuals.

7.12 Command FTESt

FTESt

FTESt for a set of linear functions or contrasts of the fixed parameters as specified by <*contrasts column*> {the number of such contrasts being <*value*>} Display on the screen the values of a set of *r* linear functions of the fixed parameters, the estimated standard errors of these values, 95% confidence limits for each value separately and simultaneously, and a joint (Wald) test for a set of *r* hypotheses, one for each value. Suppose there are *p* fixed parameters. Then the *r* hypotheses are first expressed in the form $Ax=b$ where x is the p-by-1 vector of the parameters, A is an r-by-p matrix of multipliers of the parameters such that Ax is the required set of *r* linear functions of the parameters, and b is the r-by-1 vector of hypothesised values of these functions. Thus the first row of A and the first element of b define the first hypothesis; and so on. <*contrasts column*> contains the elements of $[A,b]$ stored row by row. Thus the first $p+1$ elements of <*contrasts column*> are the first set of multipliers followed by the hypothesised value of the first linear function; and so on. <*contrasts column*> contains $r*(p+1)$ values. For example, if we have five fixed parameters and we wish to form a function which is the difference between the first two of them and another function which is the difference between the third and the fourth, and to test whether these differences are significantly different from zero, we may proceed as follows. <*value*>=*r*=2, *p*=4; let the first row of A be (1 -1 0 0 0) and the second row be (0 0 1 -1 0); b=(0 0)'. Thus we set <*contrasts column*> to (1 -1 0 0 0 0 0 0 1 -1 0 0). Note that the 5th parameter, although not part of the contrasts, still occupies spaces in <*contrasts column*>.

7.13 Command IDENtifiers

IDENtifiers

IDENtifiers for units at level <*value*> are in <*ID column*> {and for units at level <*value*> are in <*ID column*> ...} Declare column(s) containing unit identifiers at the specified level(s). The values in all ID columns must be sorted. Note that if you attempt to declare unit identifiers for a level above the maximum in force for the current worksheet the error message `Wrong parameters` will appear.

IDEN <value> (with no <*ID column*>) Remove the identification for level <*value*>.

7.14 Command LGRId

LGRId

LGRId produce a likelihood grid for random parameter(s) numbered <*value-1*> {<*value-2*> ...} taking values generated by looping through <*input-column-1*> {<*input-column-2*> ...}, deviances output to <*deviance column*> {,parameter combinations to <*output-column-1*> {<*output-column-2*> ...}} The number of input columns must match the number of random parameters specified. A value of the deviance (-2*loglikelihood) is generated for each combination of values that can be produced from the input columns. For example, if C10 = (43, 43.5, 44), C11 = (-1.19, -1.21), C12 = (0.034, 0.035) LGRId 1 2 3 C10-C12 C13 C14-C16 will generate 3*2*2=12 deviances in C13, corresponding to 3 different values of parameter 1 combined with 2 different values each of parameters 2 and 3. These 12 parameter combinations will be output as separate rows of the group C14-C16 thus:

C14	C15	C16									
43	-1.19	0.034	43	-1.19	0.035	43	-1.21	0.034	43	-1.21	0.035
43.5	-1.19	0.034	etc								

7.15 Command LIKElihood {to <box>}

LIKElihood {to <box>}
Display the value -2 * LOGE (likelihood for current model) and optionally store in <*box*>. Differencing these values allows the calculation of the deviance.

7.16 Command MAXIterations

MAXIterations <value> in batch mode.
Specify the total number of iterations, from the start, before estimation halts when in batch mode. Default is 20.

7.17 command Merge to level 1

LEV1 - merge to level 1
The lev1 operator is useful as a quick way to merge things (eg columns of higher level residuals to level 1). Graphical filtering and exclusion from model operate with columns of length number of level 1 units. Thus if we wanted to highlight all data from level 2 units with a high intercept and low slope
Calc c96= (lev1('intresid') > 1 & lev1('sloperesid') < -1) + 1
The text :
(lev1('intresid') > 1 & lev1('sloperesid') < -1)
is a logical condition returning 0 or 1. Assuming c95 is the current graph highlight column. Remembering code 0=omit from graph, code 1 = draw in normal style and codes 2-17 are highlight styles 1-16, we need to add 1 to the logical function to get the desired behaviour

7.18 Command METHod

METHod {<value>}
If <*value*>=0 set estimation method to RIGLS. If <*value*>=1 set estimation method to IGLS (the default setting). If <*value*> is absent, alternate between IGLS and RIGLS.

7.19 Command NEXT

NEXT
Continue the estimation of the current model from the current estimates.

7.20 Command OFFSet

OFFSet at level <value> from <offset column>
If <*offset column*> contains the vector *v* then *vv'* is subtracted from the residual crossproduct matrix for each block at level <*value*> in the formation of the random parameters.
OFFSet (with no parameters) Display all active offsets on the screen. OFFSet <*value*> Remove any offsets at level <*value*> which are currently linked to the model.

7.21 Command OLSEstimates

OLSEstimates
OLSEstimates for all explanatory variables in model, output the results to the screen.
OLSEstimates for each level-<*value*> unit, for all explanatory variables, results to the screen.
OLSEstimates for each level-<*value*> unit, fitting explanatory variables in <*explanatory*

variable group>, output the results to the screen. The following variations of the OLSE command have no screen output: **OLSE**stimates for each level-<*value*> unit fitting all explanatory variables, predicted values to <*predicted value column*> {coefficient estimates, 1 row for each unit, output to <*coefficient group*>} **OLSE**stimates for each level-<*value*> unit fitting explanatory variables in <*explanatory variable group*>, predicted values to <*predicted value column*> {coefficient estimates, 1 row for each unit, output to <*coefficient group*>}

7.22 Command PREDicted values

PREDicted values

PREDicted values from fixed part of current model to <*output column*> **PRED**icted values using fixed-part explanatory variable(s) <*first variable column*> {with residuals from <*first residual group*>} {<*second variable column*> {with residuals from <*second residual group*>} ...}, results to <*output column*> In the first form, all explanatory variables in the fixed part are used, with their estimated fixed coefficients excluding residuals. In the second form only those variables specified by <*first variable column*>, <*second variable column*>, etc., are used. <*first residual group*>, if specified, must contain the residuals for the coefficient of <*first variable column*> at each level that the user wishes to include in the predictions. These residuals will be added to the fixed coefficient estimate for the first variable before computing its contribution to the predicted values. And so on for <*second variable column*>, etc. For example if we wish to predict from explanatory variables C1, C2, C3 and the random coefficient of C1 has level-1 and level-2 residuals already stored in C11 and C12 then PRED C1 C11 C12 C2 C3 C4 puts into C4: the values in C1 times the sum of the fixed coefficient estimate for C1 and the level-1 and level-2 residuals for the relevant units, plus the values in C2 times the fixed coefficient estimate for C2, plus the values in C3 times the fixed coefficient estimate for C3.

7.23 Command QLIKelihood

QLIKelihood <B>

The standard command LIKE computes the likelihood for a Normal model, but will not calculate the likelihood when quasilikelihood estimation (MQL/PQL) is used. To compute the likelihood, using the Taylor series approximation and assuming Normality use the QLIK command.

QLIK B puts the estimate into box B

7.24 Command RANDom

RANDom

Display the current estimates of the random parameters.

7.25 Command RCONstraints

RCONstraints

RCONstraints put constraints on the random parameters as specified in <*constraints column*>

Suppose there are *p* random parameters and that *r* linear constraints on them are required. These are first expressed in the form *Ax=b* where *x* is the *p*-by-1 vector of the parameters, *A* is an *r*-by-*p* matrix of multipliers of the parameters, and *b* is the *r*-by-1 constrained result of the multiplication. Thus the first row of *A* and the first element of *b* define the first linear relationship; and so on. <*constraints column*> contains the elements of [*A*,*b*] stored row by row. Thus the first *p*+1 elements of <*constraints column*> are the first set of multipliers followed by the constrained value of the first linear combination; and so on. <*constraints column*> contains *r**(*p*+1) values. For example, if there are four random parameters and we wish to constrain the first three of them to be equal we may proceed as follows. *r* =2, *p*=4; let the first row of *A* be (1 -1 0 0), and the second row of *A* be (1 0 -1 0); *b*=(0 0)'. Thus we set <*constraints column*> to (1 -1 0 0 0 1 0 -1 0 0). Note that the 4th parameter, although unconstrained, still occupies spaces in the <*constraints column*>. Note also that the order of parameters is the order in which they are displayed by the RAND command. **RCON** (with no parameters) Remove any active set of constraints from the random part of the model. Alternative forms useful in macros are: **RCON** <*box*> Output the number of the current constraints column to <*box*>. If no constraints are specified, attach the last unused column on the worksheet to the model as a constraints column and return the column number <*box*>. **RCON**, add or remove according to <*value*> the constraint setting the random parameter at level <*value*> described by <*first column*> <*second column*> equal to <*value*> and output new contsraints matrix to <*new constraints column*> <*first column*>, <*second column*>, and the level <*value*> define a random parameter as in the SETE command. The constraint that this parameter should take a constant value can be either added or removed by this command. If first <*value*> = 0, remove constraint. If first <*value*> = 1, add constraint. Note: other constraints already attached to the model are included in <*new constraints column*>.

7.26 Command reflate residuals

Reflate residuals in <group> output to <group>

The <input group> contains the set of estimated at level N (determined by *MLwiN* from the length of the residuals columns) and reflates them by premultiplying by a matrix so that the reflated residuals have covariance matrix equal to the current model based estimate for that level.

We seek a transformation of the estimated residuals $\hat{U}$ of the form

$\hat{U}^* = \hat{U}A$

Write the Cholesky decomposition of *S*, the empirical covariance matrix of the estimated residuals, in terms of a lower triangular matrix as

$S = L_S L_S^T$

and the Cholesky decomposition of *R* as

$R = L_R L_R^T$

The required matrix is

$A = (L_R L_S^{-1})^T$

See also non-parametric boostrapping

7.27 Command RESEt parameters

RESEt parameters at level <value> according to <value>

During estimation a variance parameter may be estimated at a particular iteration to be

negative. This command specifies the action to be taken if this occurs. Such action may differ from level to level. The first <*value*> parameter specifies the level at which action (if any) is to be taken. The second <*value*> parameter specifies this action. If the second <*value*> parameter is 0, a negative variance estimate is reset to zero and so are any associated covariances. If the second <*value*> parameter is 1, a negative variance estimate is reset to zero but not the associated covariances. If the second <*value*> parameter is 2, no resetting takes place. Default is 0.

7.28 Command RESPonse variable

RESPonse variable is in < column>

Declare the reponse variable of the next model to be estimated.

7.29 Command RSDErrors type

RSDErrors type <value>

Set the type of standard error computation for random parameters. If <*value*> = 0, use standard, uncorrected, IGLS or RIGLS 'plug in' estimates. If <*value*> = 2, compute robust or 'sandwich' standard errors based on raw residuals.

7.30 Command RTESt

RTESt

RTESt for a set of linear functions or contrasts of the random parameters as specified by <*contrasts column*> {the number of such contrasts being <*value*>} Display on the screen the values of a set of *r* linear functions of the random parameters, the estimated standard errors of these values, 95% confidence limits for each value separately and simultaneously, and a joint (Wald) test for a set of *r* hypotheses, one for each value. Suppose there are *p* random parameters. Then the *r* hypotheses are first expressed in the form $Ax=b$ where x is the p-by-1 vector of the parameters, A is an r-by-p matrix of multipliers of the parameters such that Ax is the required set of *r* linear functions of the parameters, and b is the r-by-1 vector of hypothesised values of these functions. Thus the first row of A and the first element of b define the first hypothesis; and so on. <*contrasts column*> contains the elements of $[A,b]$ stored row by row. Thus the first $p+1$ elements of <*contrasts column*> are the first set of multipliers followed by the hypothesised value of the first linear function; and so on. <*contrasts column*> contains $r*(p+1)$ values. For example, if we have five random parameters and we wish to form a function which is the difference between the first two of them and another function which is the difference between the third and the fourth, and to test whether these differences are significantly different from zero, we may proceed as follows. <*value*>=*r*=2, *p*=4; let the first row of A be (1 -1 0 0 0) and the second row be (0 0 1 -1 0); b=(0 0)'. Thus we set <*contrasts column*> to (1 -1 0 0 0 0 0 0 1 -1 0 0). Note that the 5th parameter, although not part of the contrasts, still occupies spaces in <*contrasts column*>. The order of the random parameters is the order in which they are displayed by the RAND command.

7.31 Command SETDesign

SETDesign

SETDesign enter into the random part at level <*value*> a design vector corresponding to the data in <*column*> This command allows the user to set up a specific structure of covariances, for example an autocovariance structure. <*column*> must contain a stack of half-symmetric

matrices, one matrix for each unit at level <*value*>. *MLwiN* will use the data in <*column*> to form a vectorised block-diagonal matrix for use as a design vector to estimate a random parameter at level <*value*>. The SETDesign command may be combined with other SETDesign commands, and with SETVariances and SETElements, to specify the full covariance structure at level <*value*>. See MULSymmetric and SUBSymmetric, which may be used to set up suitable data columns for some frequently-required structures. See also CLRDesign. After using SETDesign or CLRDesign it is recommended that you continue estimation by typing STARt, not NEXT.

7.32 Command SETElements

SETElements

SETElements insert into the random part at level <*value*> the covariance matrix element(s) defined by the pair(s) of columns <*first column*> <*second column*> {<*first column*> <*second column*> ...} This command is useful when it is required to estimate some, but not all, of the variances and covariances for a set of coefficients at a particular level. Note that each covariance element to be added to the model, including a variance, requires two columns to be specified in order to define it. For example, SETE 1 C1 C2 specifies that the covariance at level 1 between the coefficients of the variables in these columns is to be estimated, but not their variances (unless these are already in the model). SETE 1 C4 C4 adds the level-1 variance of the coefficient of C4 to the model without adding its covariances with any other coefficients in the level-1 random part.

7.33 Command SETTings

SETTings {page <value>}

Display the setting screen for the current model. Each page displays five explanatory variables (if present). To see explanatory variables beyond the fifth, specify <*value*> greater than 1.

7.34 Command SETVariances

SETVariances

SETVariances and covariances at level <*value*> {for explanatory variables in <*group*>} This is the basic command for including variables in the random part of the model. Note that these must be already declared as explanatory variables. If <*group*> is not specified all declared explanatory variables are used. The command specifies that the variances and covariances at level <*value*> are to be estimated for coefficients of all the variables in <*group*>, together with their covariances with the coefficients of any other variables already in the random part at this level. The variances and covariances of the coefficients of these other variables will continue to be estimated. For example, if the level-2 random part contains just C1 as an explanatory variable (estimating just a variance for the coefficient of this variable at level 2) then SETV 2 C2 C3 will create a model in which the 3-by-3 covariance matrix of coefficients of C1, C2, C3 is estimated.

7.35 Command STARt

STARt

Start the estimation of the model currently specified. *Note that if this is in a macro you should specify 'Batch 1' if you wish to iterate till convergence.*

7.36 Command store standard errors

SEPIck standard errors

SEPIck into <C> Stacks fixed and random parts standard errors into C. Can be useful in simulations, e.g. JOIN 'res' C98 C96 'res' SEPIck C100 JOIN 'se(res)' C100 'se(res)'

7.37 Command SUMMary

SUMMary

SUMMary for level <*value*> Display a table showing, for each unit at level <*value*>, the numbers of units at lower levels that it contains, and the identifiers of units at higher levels (if any) that contain it. Unit identifier columns must have been specified previously, using the IDEN command. <*value*>, if specified, must be an integer at least 2 and not greater than the highest level in the model. If <*value*> is not specified, the highest level is assumed.

7.38 Command TOLErance <value>

TOLErance <value>

Specify the convergence criterion: if <*value*> = *m* estimation will be deemed to have converged when the relative change in the estimate for any parameter from one iteration to the next is less than $10^{(-m)}$. Default value for *m* is 2.

7.39 Command VFUN

VFUN at level N result to C [s.e. of variance function to C]

Calculates the variance function for the current model with optional standard error for the function

7.40 Command weighting mode

RWEIghting mode

RWEIght mode N Mode = 0 weighting off for residuals, mode = 1 weighting on for residuals (see WEIGhts command)

7.41 command weights

weights level N mode <M> in <C>

Mode = 1: the raw weights are in <C> Mode=2: puts standardised weights into <C>; if <C> omitted equal weights (default) are used. *MLwiN* takes the raw weights, e.g. inverse selection probabilities and standardises them so that within each higher level unit the mean weight is 1. See weighting

for further details. The following two versions of the WEIGhts command should be used to complete the specification. **WEIG**hts mode <M> Mode=0: Weighting off (equal weights) Mode=1 the specified raw weights are used at all levels Mode=2: the standardised weights are used at all levels **WEIG**hts Creates standardised weights (omit if raw weights to be used)

Commands for estimation of residuals

8.1 Command Estimation of residuals

Most of the commands in this section specify the kind of output that is required, in advance of the actual estimation. The RSETtings command displays the current specificatons, and the RESIduals command estimates and stores the residuals, and optionally their variances and covariances, according to the specifications given. The following is a logical order in which to proceed:
Decide the level at which residuals are to be estimated (the 'current level'). See RLEVel .
Decide whether you wish to output all residuals or a linear function of them, for example a prediction of the random part at the current level. See RFUNction .
Decide whether you wish to output variances and covariances. See RCOVariances .
Decide the type of variances and covariances, if any, to be output. See RTYPe.
Decide the groups which are to contain the residuals and their variances etc. See ROUTput .
Check these settings. See RSETtings.
Request the estimation and output. See RESIduals .
Residuals with missing values
To calculate interval estimates from a chain of residuals, for example from an MCMC run, see the STKRank command.

8.2 Command RCOVariances

Covariance matrices of residuals output according to <value>
If <*value*> = 0 (the default setting), output residuals only, without variances or covariances. The associated ROUTput command must specify <*residual group*> only, and not <*residual variance group*>. If <*value*> = 1, output residuals and their variances. The associated ROUTput command must specify both <*residual group*> and <*residual variance group*>, each group containing as many columns as there are residuals at the level specified by the RLEVel command, unless a linear function of residuals is requested by the RFUNction command. If <*value*> = 2, output the full covariance matrix of residuals for each unit. The covariance matrix for each unit is output as a lower-triangular matrix in stacked row order r11, r21, r22 etc., the matrices being then stacked into one column in unit order. The associated ROUTput command must specify both <*residual group*> and <*residual variance group*>. <*residual variance group*> must contain exactly 1 column, whose contents can be displayed by the MVIEw command. Variances and covariances of residuals can have one of three types. See RTYPe.

8.3 Command RESIduals

Calculate and output residuals according to current RSETttings. See RLEVel, RCOVariances, RTYPe, ROUTput, RFUNction. For example, if we have an intercept and a slope random at level 2, a suitable command sequence to calculate residuals at level 2,

together with their comparative variances, is: RLEV 2 RCOV 1 RTYP 1 ROUT C50 C51 C52 C53 RESI Residuals for the first random parameter at level 2 will be stored in C50 and, for the second, in C51; comparative variances will be stored in C52, C53. *MLwiN* automatically calculates residuals for all variables random at the specified level. The ROUTput command must specify groups with enough columns to contain these residuals, and their variances and covariances if requested. If a linear function of residuals is requested using RFUNction, only the values of this function (and optionally their variances and covariances) are output.

8.4 Command RFUNction

RFUNction; form a linear function of the residuals at the current level using <*Z group*> This command ensures that when the RESIduals command is executed, it is a linear function of residuals which is output, not the individual residuals themselves. If variances and covariances are requested by RCOVariances and RTYPe, these will be estimated for the values of the function and output to a single column. See ROUTput. The current level is specified by the RLEVel command. Suppose there are *r* variables with coefficients random at this level. Then <*Z group*> must contain *r* columns, each of length *n* where *n* is the total number of level-1 units, one column for each random coefficient. The values of the function are computed as follows. An *r*-by-1 vector *u* is formed of the residuals for the first unit at the current level. Suppose that this unit contains *m* level-1 units. Then *u* is premultiplied by the block of *m* rows of <*Z group*> which correspond to these level-1 units, to produce an *m*-by-1 vector of function values, one for each level-1 unit in the first unit at the current level. A similar calculation is made for the level-1 units in the second unit at the current level, and so on. These are the values which will be output by the RESIduals command, along with their estimated variances and covariances if requested.

8.5 Command RLEVel

RLEVel <value>
Specify the level at which residuals are to be calculated.

8.6 Command ROUTput :

ROUTput all residuals, or a linear function of the residuals, at the current level to <*residual group*> {and any required variances and covariances to <*residual variance group*>} This command specifies the groups which will be used to contain the residuals, and their variances and covariances if specified by the RCOVariances command, when the associated RESIduals command is executed. The current level is specified by RLEVel. Variances and covariances are specified by RCOVariances and RTYPe. If a linear function has not been specified by RFUNction, <*residual group*> must contain a column for each random parameter at the current level. See RCOVariances and RTYPe for the number of columns, if any, required in <*residual variance column*>. If a linear function has been specified by RFUNction, this linear function of the residuals (at the level specified by RLEVel) is computed. In this case <*residual group*> must consist of a single column, which will be used to contain the values of the function. If variances and covariances are requested by RCOVariances and RTYPe, these will be the variances and covariances of the function values. In this case <*residual variance group*> must consist of a single column, which will be used to contain these variances and covariances.

8.7 Command RSETtings

RSETtings
Display residual estimation settings.

8.8 Command RTYPe,:

RTYPe choose type of residual variances and covariances according to <*value*> If <*value*> = 0, compute diagnostic variances. If <*value*> = 1, compute comparative variances (the default setting). If <*value*> = 2, compute adjusted comparative variances. Diagnostic variances are used for standardising residuals in order to check the fit of the model. Comparative variances are used for standardising when the purpose is to compare units. Adjusted comparative variances make allowance for the fact that the random parameters, from which the residuals are computed, are themselves estimates.

8.9 Missing residuals

MISResiduals. <value >

All residual estimates that are within δ of zero are set to missing. The term δ is defined as $\mu \times 10^{-6}$ where μ is the mean of the absolute values of the raw residuals. In the predictions window, where a residual with a missing value is used in a prediction then it is treated as zero and its standard error is also set to zero.
<value>= 1 default (residuals set to missing if within δ of zero)
<value>= 0 Turn off residual missing command

Commands for model manipulation

9.1 Command ADDTerm

Create a main effect or interaction term from a series of one or more variables. Each variable can be categorical or continuous. Categorical variables can have a reference category specified, if no reference category is specified, then the lowest category number is taken as the reference category. If –1 is given as the reference category then no reference category is assumed and a full set of dummies are produced. Variables of appropriate names and data patterns are created and added as explanatory variables

9.2 Command MERGe

MERGe
MERGe using the distinct codes in <*ID-column-1*> and carrying data from <*input data group*> find corresponding codes in <*ID-column-2*> and generate corresponding data in

<output data group> The principal use of this command is to expand higher-level data into duplicate records, one record for each lower-level (typically level-1) unit. *<ID-column-1>* must be sorted and contain no duplicates. *<input data group>* must contain one row of data for each code in *<ID-column-1>*. The codes in *<ID-column-2>* can be in any order, and will typically include duplicates. For each code in *<ID-column-2>* equal to a code in *<ID-column-1>* a row is generated in *<output data group>* consisting of the data in *<input data group>* corresponding to that code in *<ID-column-1>*. For each code in *<ID-column-2>* which does not match any code in *<ID-column-1>* a row of UNKNOWN values is generated in *<output data group>*. If a code in *<ID-column-1>* is not in *<ID-column-2>* the corresponding row of data is not transferred to *<output data group>*. Thus the MERGe command can be used to remove particular units from a data set, for example to match data on new variables to existing, less complete, data. For example if level-2 units are identified in C1=(1 2 3) and we carry C2=(2.5 3.5 4.5) using a level-1-length column containing level-2 identifiers C3=(1 1 1 2 3 3) then MERG C1 C2 C3 C4 gives C4=(2.5 2.5 2.5 3.5 4.5 4.5).

9.3 Command MLAVerage

MLAVerage

MLAVerage using the codes in *<ID column>*, find the mean of the values in each column of *<input group>* for each code, output to the corresponding column in *<output group>* The codes in *<ID column>* must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of *<input group>* in turn, computes the mean value corresponding to each code in *<ID column>*, and outputs copies of these means to the corresponding column in *<output group>*, one copy for each element with the same code. Thus *<output group>* has the same number of rows and columns as *<input group>*. For example if level-2 identifiers for 7 level-1 units are in C1 = (1 1 2 2 2 3 3) and C2 = (5 6 2 4 12 2 2.5) then MLAV C1 C2 gives C3 = (5.5 5.5 6 6 6 2.25 2.25)

9.4 Command MLBOx

MLBOx

MLBOx using the codes in *<ID column>* to identify blocks of data in *<input column>*, compute five boxplot values for data in each block and store in *<output group>* The codes in *<ID column>* must be sorted, and typically represent unit identifiers at a particular level. The values computed are the minimum, lower quartile, median, upper quartile, and maximum for each block. These are stored in *<output group>* as one row for each block. *<output group>* must contain five columns.

9.5 Command MLCOunt

MLCOunt

MLCOunt using the codes in *<ID column>*, find the length of each block of identical codes and output copies of these lengths to *<output column>* The codes in *<ID column>* must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans the codes, interpreting each change of code as the end of a block. Having found the length of each block it outputs copies of these lengths to *<output column>*, one copy for each corresponding code in *<ID column>*. Thus *<output column>* has the same number of elements as *<ID column>*. For example, if level-2 identifiers are in C1 = (1 1 1 2 2) then MLCO C1 C2 gives C2 = (3 3 3 2 2)

9.6 Command MLCUmulate

MLCUmulate

MLCUmulate using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, perform a cumulative sum function for each block and output to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of <*input group*> in turn, cumulates values corresponding to each code in <*ID column*>, and outputs the results to the corresponding column in <*output group*>. Thus <*output group*> has the same number of rows and columns as <*input group*>. For example if level-2 identifiers for 7 level-1 units are in C1 = (1 1 2 2 2 3 3) and C2 = (5 6 2 4 12 2 2.5) then MLCU C1 C2 C3 gives C3 = (5 11 2 6 18 2 4.5) See also the CUMU command.

9.7 Command MLLAg

MLLAg

MLLAg using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, lag the values in each block and output to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. For example if C1 = (1, 1, 1, 2, 2, 2) and C2 = (1, 2, 3, 4, 5, 6) MLLA C1 C2 C3 produces C3 = (0, 1, 2, 0, 4, 5)

9.8 Command MLMAximum

MLMAximum

MLMAximum using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, find the maximum value in each block and output copies to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of <*input group*> in turn, selects the maximum value corresponding to each code in <*ID column*>, and outputs copies of these maxima to the corresponding column in <*output group*>, one copy for each element with the same code. Thus <*output group*> has the same number of rows and columns as <*input group*>. For example if level-2 identifiers for 7 level-1 units are in C1 = (1 1 2 2 2 3 3) and C2 = (5 6 2 12 4 2.5 2) then MLMA C1 C2 C3 gives C3 = (6 6 12 12 12 2.5 2.5)

9.9 Command MLMInimum

MLMInimum

MLMInimum using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, find the minimum value in each block and output copies to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of <*input group*> in turn, selects the minimum value corresponding to each code in <*ID column*>, and outputs copies of these minima to the corresponding column in <*output group*>, one copy for each element with the same code. Thus <*output group*> has the same number of rows and columns as <*input group*>. For example if level-2 identifiers for 7 level-1 units are in C1 = (1 1 2 2 2 3 3) and C2 = (5 6 7 4 12 2.5 2) then MLMI C1 C2 C3 gives C3 = (5 5 4 4 4 2 2)

9.10 Command MLREcode

MLREcode

MLREcode using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, assign new values starting at one for each block and output to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. <*input group*> typically consists of a single column of unit identifiers at the level below. For example if C1 = (1 1 1 1 1 1 2 2 2 2 2 2) is a column of level-3 identifiers and C2 = (2 2 2 6 6 6 4 4 5 5 7) contains the identifiers of the level-2 units within the level-3 units, MLRE C1 C2 C3 produces C3 = (1 1 1 2 2 2 1 1 2 2 3), that is, the level-2 unit identifiers have been recoded to run consecutively from 1.

9.11 Command MLSDeviation,

MLSDeviation,

MLSDeviation using the codes in <*ID column*>, compute the standard deviation of the values in each column of <*input group*> for each code, output to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of <*input group*> in turn, computes the standard deviation of values corresponding to the same code in <*ID column*>, and outputs copies of these standard deviations to the corresponding column in <*output group*>, one copy for each element with the same code. Thus <*output group*> has the same number of rows and columns as <*input group*>. For example if level-2 identifiers for 8 level-1 units are in C1 = (1 1 1 2 2 2 2 2) and C2 = (1 2 3 1 2 3 4 5) then MLSD C1 C2 C3 gives C3 = (0.817 0.817 0.817 1.414 1.414 1.414 1.414 1.414). Note that the divisor *n* is used for the s.d.

9.12 Command MLSEquence

MLSEquence

MLSEquence using the codes in <*ID column*> to identify blocks, generate sequences of numbers starting at one for each block, output to <*output column*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. The MLSEquence command typically is used to generate identifiers for the level-1 units nested within these higher-level units. For example, if level-2 identifiers for 6 level-1 units are in C1 = (1 1 1 2 2 2) then MLSE C1 C2 gives C2 = (1 2 3 1 2 3)

9.13 Command MLSUm

MLSUm

MLSUm using the codes in <*ID column*> to identify blocks of values in each column of <*input group*>, find the sum of the values in each block and output copies to the corresponding column in <*output group*> The codes in <*ID column*> must be sorted, and typically represent unit identifiers at a particular level. *MLwiN* scans each column of <*input group*> in turn, sums the values corresponding to the same code in <*ID column*>, and outputs copies of these sums to the corresponding column in <*output group*>, one copy for each element with the same code. Thus <*output group*> has the same number of rows and columns as <*input group*>. For example if level-2 identifiers for 7 level-1 units are in C1 = (1 1 2 2 2 3 3) and C2 = (5 6 2 4 12 2 2.5) then MLSU C1 C2 C3 gives C3 = (11 11 18

18 18 4.5 4.5)

9.14 Command MULSymmetric

MULSymmetric

MULSymmetric for the units defined by <*ID column*>, form a set of half-symmetric matrices of type <*value*>, by multiplying corresponding elements in <*column-1*> by those in <*column-2*>, output to <*output column*> This command sets up a data column of a particular type, for use as a design vector for a random parameter. <*ID column*> defines the level of the parameter: suppose this is level *h*. <*output column*> will contain a stacked series of half-symmetric cross-product matrices, one for each unit at level *h*. Each of <*column-1*> and <*column-2*> must be of length equal to the total number of level-1 units. The matrices in <*output column*> are produced by multiplying together appropriate elements of <*column-1*> and <*column-2*>, for example the entry for the *r*th row and *s*th column in the matrix for the first level-*h* unit is formed by multiplying the *r*th element of <*column-1*> by the *s*th element of <*column-2*>, and so on for further level-*h* units. The matrices are subjected to further processing according to <*value*>, which must be an integer greater than -3. If <*value*> = -2, leave the data unchanged. If <*value*> = -1, zeroise elements on the major diagonal of each matrix. If <*value*> = 0, zeroise elements not on the major diagonals, i.e. produce diagonal matrices. If <*value*> = *r* > 0, zeroise elements not on the *r*th minor diagonal of a matrix, i.e. produce lagged diagonal matrices, the diagonals in each case beginning on row *r*+1 of the matrix. Use the MVIEw command to check the data in <*output column*>. Use the SETDesign command to enter the data into the model as a design vector.

9.15 Command REALign

REALign classification N data in C1..C1 output to C2..C2

This takes output from the STRAnk command operating on the random classification *N* and realigns the columns C1…C1 by inserting missing value codes in relevant places to give new output columns C2…C2 which have length equal to the original number of units in the classification.

9.16 Command REMTerm

REMTerm C C …

Remove term for main effect or interaction derived from input columns C C, Columns are removed from explanatory variables and deleted. Note that if a continuous main effect was added with ADDTerm,, this would have referred to the users original data and the column will not be deleted from the worksheet.

REMT C C r

In multinomial or multivariate models a term can be repeated across a series of responses(or response categories). If the remove term command is suffixed with a number (r), then only the term for the stated response or response category is removed.

REMT C –1

If the REMT has a single listed column followed by –1 then the column is taken to be the explanatory variable corresponding to common coefficient column and that term is removed from the model.

9.17 Command REPEat

REPEat
REPEat <*value*> times the values in <*input group*> to form <*output group*> Each row in <*input group*> is repeated <*value*> times to form <*value*> successive rows of <*output group*>. The REPEat command is intended to be used in combination with the VECTorise command in order to produce corresponding variates. For example, if 500 children are identified by 'ID1'=(1 2 3 ... 500), with gender in 'SEX1', and 3 readings of height in 'HT1', 'HT2', 'HT3', then the commands VECTorise 3 "HT1' 'HT2' 'HT3' 'HT' 'OCC' REPEat 3 'ID1' 'SEX1' 'ID' 'SEX' produce 4 variables, each of length 1500. 'OCC' = (1 2 3 1 2 3 ...), 'ID' = (1 1 1 ... 500 500 500). 'HT' and 'SEX' are the height and gender of each child on each occasion (gender not varying across occasions).

9.18 Command STKRank

STKRank chain in C, number of iterations N, centiles L...L output columns C…C
This command takes a chain of *n* iterations for *m* parameters in a column ordered:

$p_{00}, p_{10}, p_{20}, p_{30}..p_{m0}, p_{01}, p_{11}..p_{m1}....p_{on}, p_{1n}..p_{mn}$

and calculates specified centiles, L,…L, using the (unsmoothed) rankings of each parameter.

This command is particularly useful for calculating intervals for the ranks of the residuals from an `MCMC` run. Note that to store residuals for an MCMC run select <model><**mcmc**><store residuals> from the MCMC menu structure.

The mcmc `store residuals` function produces a chain of residuals that does not include missing values for any missing units. Units can be missing if all the data rows for that unit contain one or more items of missing data. When we derive rank information from the residual chain it may be useful to insert missing values in the output columns from the STKRank command corresponding to any missing units. For example, if we have plotted out the ranks and their confidence intervals, we may want to identify units by clicking on them. If the plotted data set has a shorter length than the original number of relevant units (i.e. excluding missing units) the identify point routines will not form a match with a classification. This is because a point is identified as belonging to a classification if
(length of plotted data set containing point = number of units in a classification including missing units).

The output from the STKRank command can be realigned to contain missing units by the `REALign command`

9.19 Command SUBSymmetric,

SUBSymmetric,

SUBSymmetric for the units defined by <*ID column*>, form a set of half-symmetric matrices of type <*value*>, by subtracting corresponding elements in <*column-1*> from those in <*column-2*>, output to <*output column*> This command sets up a data column of a particular type, for use as a design vector for a random parameter. <*ID column*> defines the level of the parameter: suppose this is level *h*. <*output column*> will contain a stacked series of half-symmetric cross-product matrices, one for each unit at level *h*. Each of <*column-1*> and <*column-2*> must be of length equal to the total number of level-1 units. The matrices in <*output column*> are produced by subtracting elements of <*column-1*> from elements of <*column-2*>, for example the entry for the *r*th row and *s*th column in the matrix for the first level-*h* unit is formed by subtracting the *r*th element of <*column-1*> from the *s*th element of <*column-2*>, and so on for further level-*h* units. The matrices are subjected to further processing according to <*value*>, which must be an integer greater than -3. If <*value*> = -2, leave the data unchanged. If <*value*> = -1, zeroise elements on the major diagonal of each matrix. If <*value*> = 0, zeroise elements not on the major diagonals, i.e. produce diagonal matrices. If <*value*> = *r* > 0, zeroise elements not on the *r*th minor diagonal of a matrix, i.e. produce lagged diagonal matrices, the diagonals in each case beginning on row *r*+1 of the matrix. Use the MVIEw command to check the data in <*output column*>. Use the SETDesign command to enter the data into the model as a design vector.

9.20 Command SURVival times

SURVival times

SURVival times in <*time column*>, censored flags in <*censored column*>, input data in <*input data group*>, response to <*response column*>, number of failures in interval to <*failures column*>, risk set indicators to <*RSI column*>, risk set time to <*RST column*>, risk set size to <*RSS column*>, carried data to <*output data group*> Convert a set of survival times, censored flags and input data for individuals into a form suitable for survival analysis with *MLwiN*.

9.21 Command TAKE

TAKE

TAKE the first code in each block in <*input ID column*> {and the corresponding row of data in <*input group*>} and store the code in <*output ID column*> {and the corresponding row of data in <*output group*>} The codes in <*input ID column*> must be sorted, and typically represent unit identifiers at a particular level. <*input group*> and <*output group*> must both be present or both be absent. <*output ID column*> will contain the first code from each block in <*input ID column*> <*output group*>, if present, will contain the corresponding data records from <*input group*>. For example if level-2 identifiers for 5 level-1 units are in C1 = (1 1 1 2 2) with corresponding data in C2 = (1.5 1.6 1.7 2.1 2.2) then TAKE C1 C2 C3 C4 gives C3 = (1 2) and C4 = (1.5 2.1)

9.22 Command VECTorise

VECTorise

VECTorise the <*value*> variates in <*input group*> to form <*output column*> {with indicators to <*indicator column*>} <*value*> must match the number of columns in <*input*

group>. The items in the first row of <*input group*> will be placed one under the other in <*output column*>. They will be followed by the items in the second row of <*input group*>, and so on through all the rows. Thus <*output column*> has <*value*> times as many items as each variate in <*input group*>. <*indicator column*>, if specified, will contain for each item in <*output column*> the variate number within <*input group*>, starting at 1, from which the item came. For example, if <*input group*> consists of three readings of height 'HT1', 'HT2', 'HT3', each on the same 500 children, then VECTorise 3 'HT1' 'HT2' 'HT3' 'HT' 'OCC' produces a single variable 'HT', of length 1500, consisting of the 3 heights of the 1st child, followed by the 3 heights of the 2nd, and so on. 'OCC', also of length 1500, holds the numbers (1 2 3 1 2 3...) If in this example a further group contains readings for age 'AGE1', 'AGE2', 'AGE3', then VECTorise 3 'AGE1' 'AGE2' 'AGE3' 'AGE' produces the corresponding 'occasion-level' variable. See also REPEat

Commands for worksheet management

10.1 Command CTON

CTON C

turns a categorical variable to a numerical variable(that is deletes any look category name lookup info so that basic numbers are just displayed)

10.2 Command NTOC

NTOC C1

This turns a numeric variable into a categorical variable. Thus if C1 is called school and contains :

10
10
3
20
3

then C1 is turned into a categorical variable with the following number->string mapping

10 -> SCHOOL_10
3->SCHOOL_3
20->SCHOOL_20

10.3 Command FILL

FILL command - create columns length 1
FILL C-C Creates columns C-C of length 1. This is useful in conjunction with the LINK N G

which takes N empty columns from the end of the worksheet. Thus is you want 5 to reserve 5 columns in G1 and a further different 5 columns in G2 for sunsequent use in a macro then LINK 5 G1 FILL G1 LINK 5 G2 Will prevent G2 from containing the same columns as G1.

10.4 Command CTOG

CTOG Create group from columns
CTOG C G create group containing column numbers listed in C

10.5 Command ERASe

ERASe <group>
Erase the data and remove the names from all columns in <*group*>.

10.6 Command INITialise

INITialise {<value-1> {<value-2> {<value-3> {<value-4> {<value5>}}}}}
Set and display worksheet capacities for the current *MLwiN* session according to the value(s) specified. If no value is specified, display the current capacities. The capacities that can be specified are <*value-1*> number of levels (default is 5) <*value-2*> worksheet size in thousands of cells (default is 250) <*value-3*>number of columns (default is 400) <*value-4*> number of explanatory variables (default is 150) <*value-5*> number of group labels (default is 20) *MLwiN* interprets each value parameter that you specify according to its position in the list. If you wish to specify a new capacity for explanatory variables, for example, you must also specify the number of levels, the worksheet size and the number of columns (perhaps using the default values).

10.7 Command LINK

LINK <column> {<column> ...} into a group <G>
Assign a label to a group of columns. The label <*G*> must be one of G1, G2, G3, ... up to the limit in force for the current worksheet (default C20). Note that you can still refer to the individual columns of <*G*> by name (if a name is attached) or by column number. In addition, you can write <*G*>**[**<*value*>**]** to refer to a logical column within <*G*>. For example LINK C2 C7 C5 G1 AVER G1[2] will display summary statistics for column C7. An alternative format useful in macros is: **LINK** <*value*> <*G*> If <*value*> is positive, form a group with this number of columns taken from the end of the worksheet. If <*G*> already contains <*value*> columns do nothing. If <*value*> = -1 list all explanatory variables to <*G*>. If <*value*> = -2 list all fixed variables to <*G*>. If <*value*> = -3 list all fixed variables and variables random at levels above the bottom level to <*G*>. If <*value*> = -4 list, in order, the unit ID columns (highest level first), response column, explanatory variables, and, if present, offsets, weights, random constraint and fixed constraint columns (i.e. all columns associated with the current model) to <*G*>. If <*value*> < -10 list all variables random at level -(<value>+10) to <*G*>.

10.8 Command MARK

MARK mode <value> <group>
Protect the data in <*group*> from being over-written, or unprotect them, according to <*value*>. If you attempt to write into a marked group a warning will be issued asking for confirmation. If <*value*> = 0, unprotect the data in <*group*>. If <*value*> = 1, protect the data in <*group*>. **MARK** <value> If <value> = 0 do not issue overwrite warnings for marked columns If <value> = 1 do issue overwrite warnings for marked columns **MARK** (with no parameters) Display column numbers of all marked columns.

10.9 Command MOVE

MOVE columns to close gaps in sequence of columns.
Data are moved to lower-numbered columns if empty ones exist, preserving the order of the columns. A protected column (see MARK command) is not moved, nor are any data in columns numbered higher than a protected column. Since C96 is protected once a model has been run, this means that no data in columns numbered higher than C96 will be moved. For example, if C1, C2, C4, C6, C7, C9-C20 contain data, C3, C5, and C8 are empty, and none of C1-C20 is protected, MOVE will move the data in C4 to C3, C6 to C4, C7 to C5, and C9-C20 to C6-C17. C18-C20 will be empty, unless there are data further up the worksheet and no protected columns in between. If, however, C7 is protected, C4 will be moved to C3 and C6 to C4, but no data from C7 onwards will be moved.

10.10 Command NAME

NAME <C> <name> {<C> <name> ...}
Assign names of up to 8 characters to columns. **NAME**s (with no parameters) Display a summary of the names and contents of the columns in the worksheet.

10.11 Command RETRieve a worksheet

RETRieve a worksheet from a file
Respond <*filename*> to prompt. Retrieve a previously saved worksheet. If the current capacity of your worksheet is insufficient you will be warned and invited to extend it. See the INITialise command for a list of worksheet capacities and their defaults.

10.12 Command SAVE a worksheet

SAVE a worksheet {in format <value>}
Respond <*filename*> to prompt. If <*value*> is absent (the usual case), save as *MLwiN* worksheet. If <*value*>=0, save as *MLn* worksheet.

10.13 Command TIDY the worksheet

TIDY the worksheet
Clean up worksheet for more efficient use.

10.14 Command WIPE all data from the worksheet

WIPE all data from the worksheet
This command will also clear the model specification and the residual settings. The current

worksheet capacities remain in force (see INITialise command), as do the FSETtings (see FSETtings command). If logging is on (see LOGOn command) it remains on and a warning is displayed.

Commands for Multiple membership models

11.1 Command ADDM

ADDM

ADDM <value> sets of indicators at level <value>, first set in <group>, second set in <group> ….., constraints to <column>

11.2 Command Multiple membership models

There are two new commands which together can be used for specifying multiple membership models. These are WTCOl and ADDM . For details of these models with an example see Hill and Goldstein (1997).
To show the use of the WTCOl command, suppose we have a model with pupils nested within schools and we have only one response per pupil. However, some pupils attend more than one school during the study and we know the identities of the schools they attended and have some information on how much time they spent in each school.
This model is rather like a cross-classified model. We create a set of indicator variables one for each school. Where a pupil attends more than one school they require the indicator variable for each school they attended to be multiplied by a weight, which for example could be based upon the proportion of time the pupil spent at that school. The indicator variables for all the schools the pupil did not attend are set to zero. It is this set of weighted indicator variables that is made to have random coefficients at level 2. As with cross classified models, level 2 is set to be a unit spanning the entire data set and the variances of all the indicator variable coefficients are constrained to be equal.
The WTCOl command can be used to create the weighted indicator variables. If we have 100 schools and the maximum number of schools attend by a pupil is 4 then the WTCOl command would be EST".
To do this :

```
LINK C101-C200 G1              [put original indicators in group 1]
LINK C201-C300 G2              [set up group for interactions]
CALC G2=G1*'PRETEST'       [create interactions]
```

ADDM 2 sets of indicators at level 2, first set in C101-C200, second set in c201-c300, constraints to c20
This last command will place the two sets of indicators in the random part of the model as well as associated covariance terms between the two sets and establish the appropriate constraints in C20. Note that the ADDM command will not add its constraints to any existing constraints in the model

11.3 Command WTCOl

WTCOl

WTCOl <value> id columns <group 1> weights in columns <group 2> weighted indicator columns to <group 3>

Commands for cross classifications

12.1 Command A multiway cross classified example

The commands described above can also be used to model multi-way classifications. For example, our secondary school by primary school cross-classification could be further crossed by neighbourhoods. There is no neighbourhood information in this data set, but for illustration we shall simulate a neighbourhood effect and then conduct an analysis.
To run the models in this section you will need to be able to allocate a worksheet of at least 350,000 cells.
We shall simulate a structure where we have 20 neighbourhoods which introduce a random effect distributed $N(0, 0.4)$. The data are already divided into six separated secondary/primary school groups. Suppose our neighbourhoods intersect with these groups in the following way:

group	neighbourhoods contained in group
1	1-5
2	6-10
3-4	11-17
5-6	18-20

In other words, when our neighbourhoods are defined, we should be able to find 4 separate primary school/secondary school/neighbourhood groups. To simulate these data we first generate the neighbourhood codes according to the above table :

```
URAN 10000 C100
CHAN 1 C13 -1 C50
CHAN 2 C50 -6 C50
CHAN 3 4 C50 -11 C50
CHAN 5 6 C50 -18 C50
NAME C50 'BASE'
CHAN 1 C13 -4 C51
CHAN 2 C51 -4 C51
CHAN 3 4 C51 -6 C51
CHAN 5 6 C51 -2 C51
CALC C50=-C50
CALC C51=-C51
NAME C51 'RANGE'
URAN 3267 C30
CALC C15 = ROUND('BASE' + 'RANGE'*C30)
NAME C15 'NID'
```

Then we pick 20 neighbourhood effects, merge them back onto the neighbourhood codes we have just generated, and add the effects to the response :

```
CALC B1 = 0.4^0.5
NRAN 20 C31 0 B1
GENE 20 C32
MERG C32 C31 C15 C33
CALC 'ATTAIN' = 'ATTAIN'+C33
```

Before setting up the new model we clear any explanatory variables relating to the previous cross classification:

```
EXPL 0 C101-C108
```

Now we search the primary/secondary school groups for their intersections with neighbourhoods:

```
XSEA C13 'NID' C16 C17
NAME C16 'L3ID' C17 'NCODES'
```

If our simulation has worked we should find four separate groups in C16. We search these 4

new groups to create the new identifying codes for secondary schools. (Since the group structure has changed our existing identifying codes for secondary schools are out of date.)

```
XSEArch 'L3ID' 'SID' C18 C19
NAME C19 'SCODES'
```

We now re-sort the data and set up the model :

```
SORT 2 C18 C3 C1 C2 C4-C11 C13-C17 C19 C18 C3 C1 C2 C4-C11 C13-C17 C19
IDEN 3 'L3ID'
SETX 'CONS' 3 'SCODES' C101-C108 C20
SETX 'CONS' 3 'NCODES' C111-C117 C20
```

Running the model produces the results:

Parameter	Description	Estimate(se)
σ^2_{uj}	between primary school variance	1.08(0.20)
σ^2_{uk}	between secondary school variance	0.41(0.21)
σ^2_{ul}	between neighbourhood variance	0.44(0.18)
σ^2_{e}	between individual variance	8.08(0.2)
α	mean achievement	5.32(0.24)

We can model an *n*-way classification by repeated use of the XSEArch command to establish a separated group structure and then repeated use of the SETX command to specify each classification.

12.2 Command An example of a 2-way cross classification

In this example we analyse children's overall exam attainment at age sixteen - the worksheet is available on the MLwiN web site. The children are cross-classified by the secondary school and the primary school that they attended. The model is of the form given in equation (1) where u_j is a random departure due to secondary school and u_k is a random departure due to primary school. The data are on 3,435 children who attended 148 primary schools and 19 secondary schools in Fife, Scotland.

$$y_{i(jk)} = \alpha + u_j + u_k + e_{i(jk)} \qquad (1)$$

The initial analysis requires a worksheet size of 850,000 cells. Make sure that the worksheet size is large enough by referring to the **worksheet** screen on the **options** menu.

Retrieve the worksheet XC.WS. From the names menu we see that the worksheet contains 11 variables:

`VRQ` A verbal reasoning score resulting from tests pupils took when they entered secondary school.
`ATTAIN` Attainment score of pupils at age sixteen.
`PID` Primary school identifying code
`SEX` Pupil's gender
`SC` Pupil's social class
`SID` Secondary school identifying code
`FED` Father's education
`CHOICE` Choice number of secondary school attended
`MED` Mother's education
`CONS` Constant vector
`PUPIL` Pupil identifying code

Assume that a 2-level variance components model with primary school at level 2 is already set up. To add the secondary school cross-classification we need to create a level-3 unit spanning the entire data set, create the secondary school dummies, enter them as random parameters at level 3 and create a constraint matrix to pool the 20 separate estimates of the secondary school variances into one common estimate. Apart from declaring the third level which is achieved by typing

```
IDEN 3 'CONS'
```

the remaining operations are all performed by the SETX command.

The pseudo-level introduced to accommodate the cross-classification is given next. In our case this is level 3. Then comes the column containing the identifying codes for the non-hierarchical classification, which in our case is SID.

The SETX command requires the identifying codes for the cross-classified categories to be consecutive and numbered from 1 upwards. If this is not the case identifying codes can be put into this format using the MLREcode command, for example:

```
MLREcode 'CONS' 'SID' 'NEWSID'
```

Our secondary and primary identifying codes are already in the correct format so we do not need to use the MLREcode command.

The dummies for the non-hierarchical classification are written to the next set of columns. The dummies output are equal in number to *k***r* where *k* is the number of variables in <*explanatory variable group*> and *r* is the number of identifying codes in <*ID column*>. They are ordered by identifying code within explanatory variable. The constraint matrix is written to the last column. In addition the command sets up the appropriate random parameter matrix at level 3.

To set up our model the command is

```
SETX 'CONS' 3 'SID' C101-C119 C20
```

If you examine the setting screen and the worksheet you will see that the appropriate structures have been created. Before running the model we must activate the constraints by typing

```
RCON C20
```

The model will take some time to run. Four iterations are required to reach convergence and on a 90-MHz pentium each iteration takes 10 seconds. The results are:

Parameter	Description	Estimate(se)
σ^2_{uj}	between primary school variance	1.12(0.20)
σ^2_{uk}	between secondary school variance	0.35(0.16)
σ^2_e	between individual variance	8.1(0.2)
α	mean achievement	5.50(0.17)

This analysis shows that the variation in achievement at age sixteen attributable to primary school is three times greater than the variation attributable to secondary school. This type of finding is an intriguing one for educational researchers and raises many further issues for study.

Explanatory variables can be added to the model in the usual way to attempt to explain the variation. We must be careful if we wish to create contextual secondary school variables using the ML** family of commands. The data currently are sorted by primary school not secondary school as the ML** commands require. Therefore the data must be sorted by secondary school, the contextual variable created, and the data re-sorted by primary school.

Other aspects of the SETX command

When more than one coefficient is to be allowed to vary across a non-hierarchical classification in some circumstances you may not wish the covariance between the

coefficients to be estimated. This can be achieved most easily by using two successive SETX commands. The following three examples illustrate.
Example 1
```
SETX 'CONS' 3 'SID' C101-C119 C20
```
This sets up the data for the cross-classified variance components model we have just run.
Example 2
Assuming the model is set up as in example 1 and the constraint matrix is activated, if we now type
```
SETX 'VRQ' 3 'SID' C121-C139 C20
```
we shall have the structure for estimating the variance of the coefficients of VRQ and CONS across secondary schools. The intercept/slope covariance will not be estimated.
Example 3
If no cross-classified structure has yet been specified and we type
```
SETX 'CONS' 'VRQ' 3 'SID' C101-C119 C121-C139 C20
```
we shall have the structure for estimating the variance *and covariance* of the coefficients of VRQ and CONS across secondary schools.
The SETX command adds the constraints it generates to any existing constraints specified with the RCON command. Any additional random-parameter constraints which the user wants must be activated by RCON before issuing any SETX command. In particular, when elaborating cross-classified models with more than one SETX command, you must be sure to activate the constraint column generated by the first SETX before issuing second and subsequent SETX commands. Failure to do so will cause the first set of constraints not to be included in the constraints output by the second SETX command.
One limitation of the SETX command is that it will fail if any of the dummy variables it generates are already in the explanatory variable list. One situation where this may occur is when we have just estimated a cross-classified variance components model and we wish to expand the model to a cross-classified random coefficient regression with slope/intercept covariances estimated. In this case typing
```
SETX 'CONS' 'VRQ' 3 'SID' C101-C119 C121-C139 C20
```
will produce an error message since C101-C119 will already be in the explanatory variable list. The problem can be avoided by removing C101-C119 and then giving the SETX command :
```
EXPL 0 C101-C119
SETX 'CONS' 'VRQ' 3 'SID' C101-C119 C121-C139 C20
```
Note that if the random constraint matrix is left active, the above EXPL 0 command will remove from the matrix the constraints associated with C101-C119, leaving only those which the user had previously specified.
After a SETX command, estimation must be restarted using STARt. The NEXT command cannot immediately follow a SETX command.

12.3 Command BXSEarch

Note that the BXSEarch command is not available after release 1.01 of MLwiN .
BXSEarch for separable groups in the cross-classification defined by <*column-1*> and <*column-2*> putting separated group codes in <*group ID column*> and new categories in <*new ID column*>, using size limit <*value*> and method 0 {, output costs to <*costs column*>} **BXSE**arch for separable groups in the cross-classification defined by <*column-1*> and <*column-2*> putting separated group codes in <*group ID column*> and new categories in <*new ID column*>, using size limit <*value*> and method **1**, output seed to <*box*> {, output costs to <*costs column*>}

12.4 Command How cross classified models are implemented

Suppose we have a level-2 cross-classification with 100 schools drawing pupils from 30 neighbourhoods. If we sort the data into school order and ignore the cross-classification with neighbourhoods, the schools impose the usual block-diagonal structure on the *N* by *N* covariance matrix of responses, where *N* is the number of students in the data set. To incorporate a random neighbourhood effect we must estimate a non-block-diagonal covariance structure.

We can do this by declaring a third level in our model with one unit which spans the entire data set. We then create 30 dummy variables one for each neighbourhood and allow the coefficients of these to vary randomly at level 3 with a separate variance for each of our 30 neighbourhoods. We constrain all 30 variances to be equal.

We can allow other coefficients to vary randomly across schools by putting them in the model as level-2 random parameters in the usual way. If we wish the coefficient of a covariate - a slope - to vary randomly across neighbourhoods the procedure is more complicated. We must create 30 new variables which are the product of the neighbourhood dummies and the covariate. These new variables are set to vary randomly at level 3. If we wish to allow intercept and slope to covary across neighbourhoods, we require 90 random parameters at level 3: an intercept variance, a slope variance and an intercept/slope covariance for each of the 30 neighbourhoods. As before we constrain the intercept variances, the covariances and the slope variances to produce 3 common estimates by the use of appropriate constraints. The SETX command is provided to automate this procedure.

It is important to realise that although in this example we have set up a 3-level *MLwiN* structure, conceptually we have only a 2-level model, but with neighbourhood and school crossed at level 2. The third level is declared as a device to allow us to estimate the cross-classified structure. The details of this method are given in Rasbash & Goldstein (1994).

Unit IDs as categorical variables

To properly support cross classified models we need to update the way classification information is stored. With purely hierarchical models the 10^{th} entry in a school residual column corresponds to the 10^{th} school in the data set. If school is now a crossed classification and the data is not sorted according to school, we need additional information to unambiguously identify unit specific data such as residuals. To do this when cross-classified models are fitted, unit ID's are turned into categorical variables.

Thus if we have

school
10
10
...
20
10
5
30
30
5
5

this gets turned into

catname	catnum	seqnum

school_5	5	0
school_10	10	1
school_20	20	2
school_30	30	3

This information is then used by the predict and identify points facilities so that they operate correctly with cross-classified structures. The hierarchy viewer resorts to a simple summary of the number of units in each classification when in cross-classified mode.
Users should be aware that in cross-classified models all higher level ID's get categorical information attached to them, so that they appear as school_5 etc in the data window.

Some computational considerations

size of the largest group, say 1800. This leads to storage requirements of:
*3*12*1800+1800+4(1800+1800+12*1800)+2*1800*12*
that is, about 210,000 free worksheet cells.
Finding such groupings not only decreases storage requirements but also significantly improves the speed of estimation. The commands XSEArch and BXSEarch are designed to find such groups in the data.
Note that the BXSEarch command is not available in release 1.00 and later of MLwiN .

12.5 Command Modelling random cross-classifications

The motivation for multilevel modelling is that most social processes we wish to model take place in the context of a hierarchical social structure. But the assumption that social structures are purely hierarchical is often an over-simplification. People often belong to more than one grouping at a given level of a hierarchy and each grouping can be a source of random variation. For example, in an educational context both the neighbourhood a child comes from and the school a child goes to may have important effects. A single school may contain children from many neighbourhoods and different children from any one neighbourhood may attend several different schools. Therefore school is not nested within neighbourhood and neighbourhood is not nested within school: we have a cross-classified structure. The consequences of ignoring an important cross-classification are similar to those of ignoring an important hierarchical classification.
A simple model in this context can be written:

$$y_{i(jk)} = \alpha + u_j + u_k + e_{i(jk)} \qquad (1)$$

where the achievement score $y_{i(jk)}$ of the *i*th child from the (*jk*)th school/neighbourhood combination is modelled by the overall mean α, together with a random departure u_j due to school *j*, a random departure u_k due to neighbourhood *k*, and an individual-level random departure $e_{i(jk)}$.
The model can be elaborated by adding individual-level explanatory variables, whose coefficients may also be allowed to vary across schools or neighbourhoods. Also, school or neighbourhood level variables can be added to explain variation across schools or neighbourhoods.

Another example in the same context occurs when each pupil's exam paper is assessed by a set of raters. If a different set of raters operates in each school we have a pupil/rater cross-classification at level 1 nested within schools at level 2. A simple model for this situation can be written:

$$y_{(ij)k} = \alpha + u_k + e_{ik} + e_{jk} \qquad (2)$$

where the rater and pupil effects are modelled by the level-1 random variables e_{ik} and e_{jk}. (The cross-classification need not be balanced, and some pupils' papers may not be assessed by all the raters.)

If the same set of raters is used in different schools then raters are cross-classified with schools. An equation such as (1) can be used to model this situation, where now *k* refers to raters rather than neighbourhoods. If in addition schools are crossed by neighbourhoods, then pupils are nested within a three-way rater/school/neighbourhood classification. For this case we may extend equation (1) by adding a term u_l for the rater classification:

$$y_{i(jkl)} = \alpha + u_j + u_k + u_l + e_{i(jkl)} \qquad (3)$$

If raters are not crossed with schools, but schools are crossed with neighbourhoods, a simple formulation might be:

$$y_{(ij)(kl)} = \alpha + u_k + u_l + e_{i(kl)} + e_{j(kl)} \qquad (4)$$

where now *i* refers to pupils, *j* to raters, *k* to schools, and *l* to neighbourhoods.

Other applications are found, for example, in survey analysis where interviewers are crossed with areas.

12.6 Command Reducing storage overheads

We can increase speed and reduce storage requirements by finding separate groups of secondary/primary schools as described above. The XSEArch command will do this. It has the following format:

Retrieve the original worksheet in XC.WS. Readers who were unable to run the model of the previous section because of insufficient memory can start modelling at this point. We search the data for separated groups by typing

```
XSEArch 'PID' 'SID' C13 C14
```

Looking at C13, the column of separated groups produced, we see that it is a constant vector. That is, no separation can be made and all primary and secondary schools belong to one group. The new category codes in C14 therefore span the entire range (1 to19) of categories in the original non-hierarchical classification. This is not surprising since many of the cells in the 143 by 19 table contain very few individuals. It is this large number of almost empty cells that makes separation impossible. In many circumstances we may be prepared to sacrifice some information by omitting cells with very few students. We can omit data for cells with less than a given number of individuals using the XOMIt command.

Note that the BXSEarch command is not available in release 1.00 or later of MLwiN .

We deal with the XOMIt command in this section and the BXSEarch command in the next section.

In our case we can omit cells containing 2 or fewer members by typing

```
XOMIt 2 C3 C6 C1-C2 C4-C5 C7-C11 C3 C6 C1-C2 C4-C5 C7-C11
```

If we now repeat the XSEArch command exactly as before, we find that C13, the group code column, has a range from 1 to 6 indicating that 6 groups have been found. The new category codes have a range from 1 to 8 indicating that the maximum number of secondary schools in any group is 8. Look at the tabulations of secondary school and primary school by group by

typing:

```
TABU 1 'SID' C13
TABU 1 'PID' C13
```

This confirms that with our reduced data set no primary school or secondary school crosses a group boundary.

We now sort the data by primary school within separated group. The group codes are now used to define a block diagonal structure for the variance-covariance matrix at level 3 which reduces the storage overhead and speeds up estimation. The following commands set up the model.

```
SORT 2 c13 c3 C1 C2 C4-C11 C14 C13 C3 C1 C2 C4-C11 C14
IDEN 3 C13
SETX 'CONS' 3 C14 C101-C108 C20
RCON C20
```

Notice that the new category codes in C14 running from 1 to 8 (the maximum number of secondary schools in a separated group) are now used as the category codes for the non-hierarchical classification. This means we now need only 8 as opposed to 19 dummies to model this classification.

Estimation proceeds more than four times faster than in the full model, with very similar results.

Parameter	Description	Estimate(se)
σ^2_{uj}	between primary school variance	1.10(0.20)
σ^2_{uk}	between secondary school variance	0.38(0.19)
σ^2_{e}	between individual variance	8.1(0.2)
α	mean achievement	5.58(0.18)

Note that the BXSEarch command has been dropped from version 1.00 onwards.

12.7 Command SETX

SETX set a random cross-classification, with coefficients of *<explanatory variable group>* random at level *<value>* across categories in *<ID column>*, storing dummies in *<output group>* and constraints in *<constraints column>* *<explanatory variable group>* specifies the variables whose coefficients we wish to vary randomly across the non-hierarchical classification, in our case, the secondary schools. Here we wish to run a variance components model so only CONS varies across secondary schools.

12.8 Command XOMIt

XOMIt cells with not more than *<value>* members from the cross-classification defined by *<input column-1>* and *<input column-2>* {carrying data in *<input data group>*} results to *<output column-1>* *<output column-2>* {and carried data to *<output data group>*}

12.9 Command XSEArch

XSEArch for separable groups in the cross-classification defined by *<column-1>* and

<column-2> putting separated group codes in *<group ID column>* and new categories in *<new ID column>* The first two columns describe the cross-classification to be searched. The non-hierarchical classification is specified by *<column-2>*. If separable groups can be found they are assigned group codes 1, 2, etc. and these are placed in *<group ID column>*. The category codes of *<column-2>* are then recoded in *<new ID column>* to run from 1 within each group. An example will make this clear.

Commands to access IGLS algorithm

13.1 Access to the IGLS algorithm

Access to the IGLS algorithm - introduction

A number of commands give access to the matrices used by the IGLS estimation algorithm of *MLwiN*. The formats of all the matrix commands are described in the matrix section of the help system. For convenience a list of these is given here. *These macros can be downloaded from the MLwiN web site:*

13.2 Command Implementing the IGLS algorithm in macros

A macro implementation of the IGLS estimation algorithm

The macros can be obtained as a zipped file from the MLwiN web site. The files are:
RUN : runs a model to convergence
ITER : performs an iteration
RAND : accumulates SSP matrices
DOXXR : a component of random part estimation
FIXED : fixed coefficients and their covariance matrix
KRON : a component of random part estimation
NAMEs : Names the columns used by the macros.
The macros provide an extended example of the use of the matrix commands of *MLwiN*. They also provide a description of the IGLS (iterative generalised least squares) estimation algorithm. The model which it is desired to estimate should be specified in the usual way. As the macros are run all internal algorithmic matrices become available for scrutiny or alteration. This allows new methodological developments to be tested conveniently on reasonably small data sets. An example is also given.

13.3 Command macro DOXXR

```
loop b70 1 b60                [loop b70 : 1->i, traverse to the diagonal]
  calc 'temp'= diag(g1[b60] *. 'vinv' *. g1[b70] *.vinv')[diag(A_i V^-1 Y V^-1)]
  sum 'temp' b62              [get trace..]
  pick b61 'xxr' b71
  calc b71=b71+b62            [..and accumulate]
  edit b61 'xxr' b71
  calc b61=b61+1
endloop
```

13.4 Command macro FIXED

```
nlev b50                      [store highest level in b50]
nunit b50 b51                 [store number of highest level units in b51]
```

```
nfix b55                          [number of fixed parameters to b55]
calc b56=b55^2
put b55 0 'xyf'                   [zero xyf]
put b56 0 'xxf'                   [zero xxf]
matr 'xxf' b55 b55                [declare xxf as a matrix]
say \n
loop b50 1 b51                    [for each block]
  say .
  ymat b50 'y'                    [get y]
  vmat b50 'v'                    [get V]
  calc 'vinv'=inv('v')            [invert V]
  xmat b50 'x'
  calc 'temp'=(~'x')*.'vinv'*.'x'
  calc 'xxf'='xxf'+'temp'         [accumulate X^T V^-1 X ]
  calc 'temp'=(~'x')*.'vinv'*.'y'
  calc 'xyf'='xyf'+'temp'         [accumulate X^T V^-1 y ]
endloop
say \n
calc c98=inv('xxf')*.'xyf'        [calculate β and assign to C98]
calc c99=hsym(inv('xxf'))         [assign covar(β) to c99]
```

13.5 Command macro ITER

macro ITER

This macro performs an iteration. It first calls NAMES so that the macro can be run independently of macro RUN.

```
obey names
obey rand
obey fixed
```

13.6 Command macro KRON

This macro evaluates for the current block

$$X^{**T}(V^{-1} \otimes V^{-1})Y^{**}$$

and calls the macro DOXXR which evaluates

$$X^{**T}(V^{-1} \otimes V^{-1})X^{**}$$

Calculation of the full Kronecker product is avoided by using the formulation

$$D = tr(AV^{-1}BV^{-1})$$

where D is an element of xxr or xyr.

In a 2-level random coefficients regression model with constant level-1 variance xxr is an r by r symmetric matrix with the following structure :

		B		
	$B_1 = x_0x_0^T$	$B_2 = x_0x_1^T + x_1x_0^T$	$B_3 = x_0x_0^T$	$B_4 = diag(x_0x_0^T)$
A	$A_1 = x_0x_0^T$	$tr(A_1V^{-1}B_1V^{-1})$		
	$A_2 = x_0x_1^T + x_1x_0^T$	$tr(A_2V^{-1}B_1V^{-1})$	$tr(A_2V^{-1}B_2V^{-1})$	
	$A_3 = x_0x_0^T$	$tr(A_3V^{-1}B_1V^{-1})$	$tr(A_3V^{-1}B_2V^{-1})$	$tr(A_3V^{-1}B_3V^{-1})$

$A_4 = diag(x_0 x_0^T)$ $tr(A_4 V^{-1} B_1 V^{-1})$ $tr(A_4 V^{-1} B_2 V^{-1})$ $tr(A_4 V^{-1} B_3 V^{-1})$ $tr(A_4 V^{-1} B_4 V^{-1})$

xyr is an *r* by 1 vector with the structure :

$tr(A_1 V^{-1} Y\ V^{-1})$

$tr(A_2 V^{-1} Y V^{-1})$

$tr(A_3 V^{-1} Y V^{-1})$

$tr(A_4 V^{-1} Y V^{-1})$

where $Y = \tilde{Y}^{**} = y\,\tilde{y}$ and *r* is the number of random parameters. For further details see Goldstein and Rasbash (1992).

Macro KRON loops through all the *A*s and calculates xyr[*i*]= $tr(A_i V^{-1} Y\ V^{-1})$. On each pass of the loop macro DOXXR is called which calculates the *i*th row of matrix xxr, omitting symmetric cells. Note that xxr is held as a column of length *r*(r+1)/2* since *MLwiN* does not support symmetric matrices.

13.7 Command macro RAND

Note: $X^{**T}(V^{-1} \otimes V^{-1})X^{**}$ = xxr, $X^{**T}(V^{-1} \otimes V^{-1})Y^{**}$ = xyr

```
nlev b50                          [store highest level in b50]
nunit b50 b51                     [store number of highest level units in b51]
nrnd b54                          [number of random parameters to b54]
calc b55=b54*(b54+1)/2            [calculate number of elements in xxr]
put b54 0 'xyr'                   [zero xyr]
put b55 0 'xxr'                   [zero xxr]
link b54 g1                       [create a group of size num params to hold X** ]
say \n
loop b50 1 b51                    [for each block]
  say .                           [send '.' to screen]

  yres b50 'YR'                   [get ỹ ]
  xss b50 g1                      [form X** ]
  count 'yr' b53                  [set b53 to block size]
  itnum 0 b52
  switch b52
    case 0 :
        note first iteration      [first iteration use I..]
        imat b53 'vinv'
        leave
    case :
        note otherwise            [..otherwise construct V]
        vmat b50 'V'
        calc 'vinv'= inv('v')     [and invert it]
  ends
  obey kron                       [accumulate xxr and xyr]
endloop
calc c96=inv(sym('xxr'))*.'xyr'   [calculate estimates and assign to c96]
calc c97=hsym(inv(sym('xxr')) * 2)  [assign covariance of estimates to c97]
note set iter. number > 0 so next call does not restart
itnum 1 1
```

13.8 Command macro RUN

This macro will run a model to convergence. If the iteration number is set to zero the macro will generate starting values, otherwise the macro will continue from whatever values are in C96 and C98.

Note - you can report or set the iteration number with the **ITNU**mber command

ITNU mode 1 <*value*> sets iteration number

ITNU mode 0 <*box*> sets box to current iteration number

```
obey names                        [name columns used by macros]
itnum 0 b52                       [set b52 to iteration number]
switch b52
  case 0:
        note zero estimates       [set c96&C98 to small values.. ]
        nrnd b54                  [..for convergence comparison]
        put b54 0.05 c96
        nfix b54
        put b54 0.05 c98
ends
```

```
loop b57 1 100          [loop for 100 iterations but quit if convergence is acheived]
  calc 'old_rp'=c96                          [record previous iterations  estimates]
  calc 'old_beta'=c98
  obey iter                                  [do an iteration]
  calc 'temp'= !(abso((c96-'old_rp')/'old_rp') < 0.01)
  sum 'temp' b58
  calc 'temp'= !(abso((c98-'old_beta')/'old_beta') < 0.01)
  sum 'temp' b59
  calc b58=b58+b59
  switch b58                                 [exit if converged]
     case 0:
        note converged
        say \n\nCONVERGED\n
        return
  ends
endloop
```

Note that the command

```
calc 'temp'= !(abso((c96-'old_rp')/'old_rp') < 0.01)
```

checks for random parameter convergence. The ! operator (logical not) is applied to the convergence vector. The convergence vector has one element for each parameter. The terms

```
abso((c96-'old_rp')/'old_rp') < 0.01
```

construct the convergence vector such that an element is 1 if the parameter's estimation has converged, 0 if it has not. This is because *MLwiN* codes 1 as true and 0 as false. If the sum of the logically negated fixed and random parameter convergence vectors is zero then convergence is achieved.

13.9 Example using the algorithm access macros

```
retr jspmat.ws
echo
fpath algor
itnum 1 0                  [reset itnum so starting values generated]
obey run
```

Because the macros update C96-C99 we can use the RAND and FIXE commands to view the estimates in the output screen of the command interface.

```
rand
LEV.  PARAMETER       (NCONV)    ESTIMATE    S. ERROR(U)  PREV. ESTIM     CORR.
-------------------------------------------------------------------------------
 2    CONS     /CONS     ( 1)       5.324          1.454         5.32         1
 2    RAVENS   /CONS     ( 1)     -0.3801         0.1293      -0.3787    -0.879
 2    RAVENS   /RAVENS   ( 2)     0.03513         0.0167      0.03503         1
-------------------------------------------------------------------------------
 1    CONS     /CONS     ( 2)       27.19          1.304        27.19         1

fixe


  PARAMETER            ESTIMATE     S. ERROR(U)   PREV. ESTIMATE
CONS                      29.74        0.422             29.74
RAVENS                   0.6133      0.04233            0.6133
SEX                      -0.2615       0.3487          -0.2617
```

The columns of previous estimates and convergence counts are not updated by the macros and should be ignored. They are only correct in this instance because a normal model had been run prior to running the macros. The rest of the display is correct.

Commands to manipulate data structures

14.1 Comamnd OMEGa

OMEGa, output omega, the random parameter covariance matrix, at level <value> to <output column>
OMEGa, output submatrix of omega at level <*value*> corresponding to explanatory variables in <*explanatory variable group*> to <*output column*>

14.2 Command BLKTotals

BLKTotals, for unit number <value>, place number of units at each level in <output column>
<*value*> refers to a unit at the highest level. <*output column*> will contain the numbers of units at each level which are contained within this highest-level unit, starting with level 1 and finishing with the highest level (for which the number of units in this unit is always 1). The unit identifiers must have been previously defined using the IDENtifiers command.

14.3 Command CHOL

CHOL
CHOL C1 C2 takes the lower triangle of a symmetric matrix (S) stored columnwise in C1 and places the cholesky decomposed matrix (L) as a square matrix into C2. See also the MATRix command.
The Cholesky decomposition is defined by
$S = L^T L$
where L is a lower triangular matrix, i.e. with zeros above the diagonal.

14.4 Command MKBLock,

MKBLock create a block-diagonal matrix (or matrices) corresponding to block <*value-1*>, with sub-blocks corresponding to level <*value-2*>, using values in <*input group*>, output to <*output group*> <*value-1*> refers to a unit at the highest level. Suppose this unit contains *m* sub-units at level <*value-2*> and *n* sub-units at level 1. Then each column in <*input group*> must contain *m* values, 1 for each sub-unit. For the first column an *n*-by-*n* block-diagonal matrix is generated, with sub-blocks for each level-<*value-2*> sub-unit. The *r*th sub-block will contain replicates of the *r*th value in the column. The block-diagonal matrix is stored in the first column of <*output group*>. And so on for any further columns of <*input group*>, generating an equal number of columns in <*output group*>. Note that the unit identifiers must have been previously defined using the IDENtifiers command.

14.5 Command MVIEw

MLwiN stores a covariance matrix as a lower-triangular matrix whose rows are stacked into a single column. For example, the variances and covariances of model parameter estimates are held in this form in columns C97 (for random parameters) and C99 (for fixed parameters). Residual covariance matrices specified by RCOV 2 are output in this form, one such matrix for each unit at the specified level, with the matrices stacked into a single column. MVIEw will display these matrices in lower-triangular form

For example, if C1 = (1 1 1 2 2 2) and C2 = (1 2 3 4 5 6 7 8 9 10 11 12),

14.6 Command SUBBlock,

SUBBlock,
SUBBlock for block <*value-1*> which is contained in <*input column*>, output the sub-block corresponding to the <*value-2*>th sub-unit at level <*value-3*> to <*output column*> <*value-1*> refers to a unit at the highest level. Suppose this has *n* sub-units at level 1. Then <*input column*> must contain an *n*-by-*n* block-diagonal matrix. The command uses the unit identifier columns previously defined as part of the model, together with <*value-2*> and <*value-3*>, to find the requisite sub-block and output it to <*output column*>.
The submatrix of V corresponding to unit number <*value*> at the highest level is output as a square matrix to <*output column*>.

14.7 Command VMATrix,

VMATrix output V, the covariance matrix of the responses, block <*value*> to <*output column*> The submatrix of V corresponding to unit number <*value*> at the highest level is output as a square matrix to <*output column*>.
V is stored as a square matrix since the CALC command has no special provision for symmetric matrices.
For example,

```
vmat 42 c100
calc g1=c100
prin g1
       5.0587
  5  3.0752        5.7198        1.7530        5.0587        40.840
calc g2=g1 *. inv(g1)
prin g2
       C395        C394         C393         C392         C391
N =       5           5            5            5            5
  1  1.0000       1.1646e-10   4.0766e-10   -7.9638e-10   3.7795e-09
  2  6.6288e-09   1.0000       4.1802e-09   -8.5903e-09   1.2766e-08
  3  -5.2018e-10  -1.8630e-09  1.0000       -2.9530e-09   -7.1374e-10
  4  3.2922e-09   -4.5401e-09  -2.4225e-10  1.0000        5.5826e-09
  5  1.0988e-08   -8.6555e-09  5.9089e-09   -1.2029e-08   1.0000
```

14.8 Command XMATrix

XMATrix, output X, the fixed-part design matrix, block <value> to <output column>
Output the matrix of fixed-part explanatory variables corresponding to unit number <*value*> at the highest level, to <*output column*> See access to the IGLS algorithm for examples of use.

14.9 Command XSS

XSS output block <value> of the design matrix for the random parameters to <output group>
The specified submatrix of X**, the design matrix for the random parameters, is output to <*output group*>. It corresponds to a unit at the highest level. <*output group*> must contain columns to match the design vectors in X**. See access to the IGLS algorithm for examples of use.

14.10 Command YMATrix

YMATrix, output the response vector for unit number <value> at the highest level to <output column>

14.11 Command YRESiduals

YRESiduals, output the raw residual vector for unit number <value> at the highest level to <output column>

14.12 Command ZMATrix,

```
 N =      5           5
   1  1.0000        8.0000
   2  1.0000        4.0000
   3  1.0000        10.000
   4  1.0000        5.0000
   5  1.0000        -8.0000
```

XSS output block <*value*> of the design matrix for the random parameters to <*output group*> The specified submatrix of X**, the design matrix for the random parameters, is output to <*output group*>. It corresponds to a unit at the highest level. <*output group*> must contain columns to match the design vectors in X**.

For example

```
link c21-c24 g1  [2 level random coeff. model so 4 output cols. needed]
xss 42 g1
prin g1
       C21          C22          C23          C24
N =     25           25           25           25
  1  1.0000        16.000        64.000        1.0000
  2  1.0000        12.000        32.000        0.0000
  3  1.0000        18.000        80.000        0.0000
  4  1.0000        13.000        40.000        0.0000
  5  1.0000        0.0000        -64.000       0.0000
  6  1.0000        12.000        32.000        0.0000
  7  1.0000        8.0000        16.000        1.0000
  8  1.0000        14.000        40.000        0.0000
  9  1.0000        9.0000        20.000        0.0000
 10  1.0000        -4.0000       -32.000       0.0000
 11  1.0000        18.000        80.000        0.0000
 12  1.0000        14.000        40.000        0.0000
 13  1.0000        20.000        100.00        1.0000
 14  1.0000        15.000        50.000        0.0000
 15  1.0000        2.0000        -80.000       0.0000
 16  1.0000        13.000        40.000        0.0000
 17  1.0000        9.0000        20.000        0.0000
 18  1.0000        15.000        50.000        0.0000
 19  1.0000        10.000        25.000        1.0000
 20  1.0000        -3.0000       -40.000       0.0000
 21  1.0000        0.0000        -64.000       0.0000
 22  1.0000        -4.0000       -32.000       0.0000
 23  1.0000        2.0000        -80.000       0.0000
 24  1.0000        -3.0000       -40.000       0.0000
 25  1.0000        -16.000       64.000        1.0000
```

The order in which the columns are used corresponds to the order in which random parameters are displayed by the `RAND` command.

We usually wish to treat the n^2 by 1 design vectors as n by n matrices. Therefore for

convenience the individual design vectors are dimensioned *n* by *n*:

```
mdim g1[1]
5 rows by 5 columns
```

Even though the components of G1 are dimensioned as square matrices G1 as an entity can still be treated as a matrix dimensioned n^2 by the number of random parameters :

```
calc g2=(~g1) *. (g1)
prin g2
        C387          C386          C385          C384
N =        4             4             4             4
  1   25.000        190.00        361.00        5.0000
  2   190.00        3412.0        10222.        38.000
  3   361.00        10222.        72361.        269.00
  4   5.0000        38.000        269.00        5.0000
```

14.13 matrices derived from a model.

Matrices derived from a model.

It is possible to have access to the matrices formed and used in (R)IGLS estimation. These can then be manipulated using the general matrix commands .

There is also a set of macros , using these matrices, which have been written to carry out detailed manipulations of the (R)IGLS algorithm.

The following matrix commands are available:

BLKTotals Extracts the total number of units at each level

MKBLock Creates block diagonal matrices containing specified values for subunits of a unit at a specified level.

MVIEw Views the lower triangles in 2 dimensions, of matrices stored in stacked format

OMEGa Obtains the random parameter covariance matrix at a specified level.

SUBBlock Obtains a specified subunit (sub-block) from a specified higher level unit.

VMATrix Outputs V, the estimated covariance matrix of total residuals.

XMATrix Outputs fixed part design matrix for a specified unit at highest level.

XSS Outputs design matrix of random parameters for a specified highest level unit.

YMATrix Outputs response vector for specified unit at the highest level.

YRESidual Outputs total (raw) residual vector for specified unit at highest level.

ZMATrix Ouputs random part design matrix for explanatory variables defined at a specified level for a specified unit at the highest level.

Macro commands

15.1 command OBPAth

OBPAth S1

Place the current fpath setting into string variable S1

15.2 Command RESUme

RESUme
Resume execution of the macro that is currently suspended by a PAUSe command. If no macro is suspended, RESUme has no effect.

15.3 Command ABORt macro execution

ABORt macro execution and return to terminal input

15.4 Command ASSIgnnumbers

ASSIgn numbers to <group>
This command provides an equivalent function to the old INPUt command which is no longer available, namely: Lines following this command should contain the numbers, separated by spaces, which are required to be stored in <*group*>. The numbers on each line will be placed one by one into successive columns of <*group*>, overwriting any pre-existing contents. It is permissible to type a line containing fewer numbers than there are columns in <*group*>, but once the last column of <*group*> has been reached the current line of numbers must end. Further numbers may be input on a new line, and will be joined one by one to the ends of the columns of <*group*>. FINIsh, or another command, terminates the input of numbers.

15.5 Command BREAk

BREAk
Exit from loop. See LOOP .

15.6 Command CALLer identifier

CALLer identifier to <box>
Put a value in <*box*> to indicate the caller of this macro. <*box*> = 1 if called by STARt or NEXT <*box*> = 2 if called by RESIduals <*box*> = 3 if called by LIKElihood <*box*> = 4 otherwise

15.7 Command CONVergence status

CONVergence status output to <box>
Status 0: estimation has not converged. Status 1: estimation has converged. Status 2: ssp matrix or V is -ve definite.

15.8 Command ENDLoop

ENDLoop
This command ends the scope of a loop. The result of its execution depends on the current values of <*box*>, <*upper value*>, and <*step value*>, as specified on the corresponding LOOP command. If <*box*> = <*upper value*>, execution of the loop ceases and control is passed to the command which follows ENDLoop. If not, <*step value*> (or the default value +1) is added to <*box*> and control is passed to the command which follows the corresponding LOOP command.

15.9 Command ERROr mode

ERROr mode <value>
If <*value*> = 0, do not abort macros if an error occurs. If <*value*> = 1 (the default setting), abort macros if an error occurs.

15.10 Command EVARiables,

EVARiables,
EVARiables output the column numbers of all explanatory variables to <*column*> {with fixed part indicators to <*indicator column*>} <*indicator column*> will contain a 1 or a 0 according to whether the corresponding explanatory variable is or is not in the fixed part of the model.

15.11 Command EXISt

EXISt, put indicator of the existence of column <name> in <box>
<*box*> is set to 1 if column <*name*> exists, otherwise set to 0.

15.12 Command FPATh

FPATh <directory>
Set a search path to the directory for macro files. If a macro file is not in the current directory it is looked for in the directory specified with the FPATh command. For example, FPATh c:\mlwin\discrete causes *MLwiN* to look for macro files in the current directory and then in c:\mlwin\discrete. **FPAT**h (with no parameters) Clear the current search path for macro files.

15.13 Command FSET

FSET
Display current FPATh, PREFile and POSTfile settings.

15.14 Command GALL

GALL <Input Column>
Changes the highlight filter column to <Input Column>
If entry in <Input Column> = 0 leave out the level 1 unit from graph
If entry = (n; n=2…,17) plot in style n-1.
Note that the graph highlight column can also be changed from the graph highlight window.

15.15 command GBARwidth

GBAR width N
GBAR N Set barwidth for current data set of current graph, only applies when data are plotted as a histogram

15.16 Command GCLEar clear all graph sets.

GCLEar clear all graph sets.
GCLEar <value> : clear all data sets in graph set <value> **GCLE**ar <value 1> <value 2>: clear the (value 1> the data set from graph set <value 2>.

15.17 Command GCOLumn

GCOLumn for level <value> to <box>
Output the column number for the unit identifiers at level <*value*> to <*box*>.

15.18 Command GCOOrdinate

Attach the current data set to the (<value 1>,<value 2>)'th sub-graph in the current table of sub-graphs. Note that the top left sub-graph has co-ordinate (0,0).

15.19 Command GDIVide

GDIVide <group> by <column>
Divide each column in <*group*> by <*column*>, results to <*group*> overwriting original contents. This command has no storage overhead.

15.20 Command GETYpe

GETYpe <value>
Set display type for Y errors. <value> = 0 draw as error bars, <value> = 1 draw as lines.

15.21 Command GFILter

GFILter <column>
Defines a filtering column for the current data set, only those data points which have a non-zero positive number in the corresponding entry in the filter column will be included in the plot.

15.22 Command GGRId

If <value> = 0 then do not show any row or column titles for the current table(or grid) of graphs. If <value> = 1 then show any row or column titles for the current table(or grid) of graphs

15.23 Command GGROup

GGROup <column>
Define a grouping variable for the current data set. Particularly useful for grouped line plots. GGROup with no parameters clears the grouping column.

15.24 Command GIBBS

GIBBS
This is now superseded by the MCMC command.

15.25 Command GINDex

GINDex

GINDex graph set <value>, data set <value> Subsequent graphics commands refer to this data set, until a new GIND command is issued.

15.26 Command GLSTyle

GLSTyle <value>

Set line style for current data set. <value> can take a value of 1 to 6. Value 1 for solid lines, values 2 to 5 for varying lengths of dash. This command only has an effect if the current data set is being plotted as a line.

15.27 Command GLTHickness

GLTHickness <value>

Set line thickness in pixels for the current data set . This command only has an effect if the current data set is being plotted as a line.

15.28 Command GMSTyle

GMSTyle <value>

Set marker(symbol) style for the current data set. <value> may take a value of 0 to 13 corresponding to 14 different symbol types. This command only has an effect if the current data set is being plotted as a points.

15.29 Command GMULtiply

GMULtiply <group> by <column>

Multiply each column in <*group*> by <*column*>, results to <*group*> overwriting original contents. This command has no storage overhead.

15.30 Command GORDer N

GORDder N

Set plot order number for current data set. Higher numbers plot last and therefore appears at the front.

15.31 command graph highlight

Graph highlight

GHIG mode N
Mode = 1 exclude current data set from graphical highlighting,
Mode = 0 include
Default is 0.

15.32 Command graph label

Graph label

GLABEl <STRING>: set text label for current data set GLABEL - clear text label for current data set GLABEL N N=0 hide label for current data set N=1 use specified label text N=2 create text from group category names(in grouped plots)

15.33 Command graph scales

Graph scales

GSCAle row R col C mode M Autoscale the graph in row R, column C of current display M=0 Y axis, M=1 X axis GSCAle row R col C mode M, min N, max N, nticks N User defined scale for the graph in row R, column C of current display M=0 Y axis, M=1 X axis Then specify maximum, minimum values for axis and number of ticks on axis.

15.34 Command graph text label

Graph text label

GTEXt mode N Show text labels assigned to data sets as : N=0 hide all text, N=1 in a single legend regardless of howmany graphs in the display N=2 in legends with one legend per graph N=3 in lables

15.35 Command GSET

GSET<value> <first G> <second G> <result G>

Perform set-algebraic operations on two existing groups to form a new group. If <*value*> = 0 link columns common to <*first G*> and <*second G*> as <*result G*> (intersection) If <*value*> = 1 link columns in <*first G*> but not in <*second G*> as <*result G*> (difference) If <*value*> = 2 link columns in either <*first G*> or <*second G*> as <*result G*> (union) For example, if G3 contains C1,C4,C5,C8 and G4 contains C5,C8,C9 GSET 0 G3 G4 G5 links C5,C8 as G5 GSET 1 G3 G4 G5 links C1,C4 as G5 GSET 2 G3 G4 G5 links C1,C4,C5,C8,C9 as G5

15.36 Command GSIZe

GSIZe, output the number of columns in <G> to <box>

15.37 Command GSSZ

GSSZ <value>

Set the symbol size for the current data set, measured in thousandth's of the overall size of the graph window. This command only has an effect if the current data set is being plotted as a points.

15.38 Command GTABle

GTABle <value 1> <value 2>
Divide the current graph set into a <value 1> by <value 2> table of sub-graphs, <value 1> rows by <value 2> columns

15.39 Command GTITle

GTITle <value 1> <value 2> "title string"<value 1> = 0, title row <value 2> of the current table of graphs with "title string" <value 1> = 1, title column <value 2> of the current table of graphs with "title string"**GTIT**le <value 1> <value 2> <value 3> "title string" <value 1> = 2, supply a title for the x-axis of the (<value 2>,<value 3>)'th subgraph in the current table of graphs <value 1> = 3, supply a title for the y-axis of the (<value 2>,<value 3>)'th subgraph in the current table of graphs <value 1> = 4, supply a graph title for the (<value 2>,<value 3>)'th subgraph in the current table of graphs For example if we have a table of graphs with only 1 sub-graph in it, which is the default and commonest case, and we wish to give the x axis of this graph the title AGE, then this is achieved by the command GTITle 2 0 0 "AGE"

15.40 Command GTYPe

GTYPe <value>
Set graph type for current data set. <value> = 0:line, 1:point, 2:line+point

15.41 Command GXCOl

GXCOl <column>
Set X data to specified column. GXCOl with no parameters clears the X data column.

15.42 Command GYCOl

GYCOl <column>
Set Y data to specified column. GXCOl with no parameters clears the Y data column.

15.43 Command GYERror

GYERror
GYERror 1 <column> set positive Y errors associated with current data set to <column>
GYERror 2 <column> : set negative Y errors associated with current data set to <column>
GYERror with no parameters clears positive and negative errors associated with current data set
GYERror 1 : clear positive errors. **GYER**ror 2 : clear negative errors

15.44 Command IDCOlumn

IDCOlumn
column numbers for the unit identifiers at all levels to <*column*>. For example, given IDEN 1 C1 2 C10 3 C20 then IDCO 2 B1 puts the value 10 in B1, IDCO C5 gives C5 = (1 10 20)

15.45 Command IMAC

IMAC <value 1>
The PAUSE statement with no parameters returns control to the front end and the macro user. To give up control at regular intervals might prove frustrating to the user as they will be continually

hitting the resume macro button. In some situations what is required is for the macro user to have more control over when a macro pauses. The macro interruption command, IMAC, can be used for this purpose. If IMAC <value 1> is set to 1 then when a macro is executing a button labelled "Pause Macro" appears on the main toolbar. If the user presses this button whilst a macro is executing then any subsequent PAUSE 1 macro statements will cause control to go the front end and the resume and abort macro buttons will appear. The front end will then respond to users in the usual way. If the resume button is pressed then the abort and resume buttons disappear, the pause button reappears and the macro resumes execution. If macro interruption is switched on and the user has not pressed the pause button on the main toolbar when a PAUSE 1 statement is executed then the macro will update the front end and continue automatically. The default setting for macro interruption is off, which corresponds to <value 1> = 0. Note that if a macro issues an IMAC 1 statement the pause button does not appear on the main toolbar until a subsequent PAUSE or PAUSE 1 statement is executed.

15.46 Command INMOdel

INMOdel, if the random parameter at level <value> defined by <column-1> and <column-2> is in the model, set <box>
<*box*> is set to 1 if the specified random parameter is in the model, else set to 0.

15.47 Command ITNUmber

ITNUmber 0 {<box>}
Display the current iteration number and store in <*box*> if specified. **ITNU**mber **1** <*value*>
Set the current iteration number to <*value*>.

15.48 Command LOOP

LOOP for <box> going from <lower value> to <upper value> {<step value>}
The default step value is 1. For example, LOOP B1 1 5 statements ENDLoop Note that no single macro file should contain more than one loop. For example, LOOP B1 1 5 LOOP B2 2 10 .... ENDL ENDL will result in errors. Code instead: LOOP B1 1 5 OBEY L2 ENDL where L2 is a macro file containing the nested loop.

15.49 Command Macro commands for graphics

Macro commands for graphics

The commands for manipulating graphics assume familiarity with the graphical interface. For a full description of this interface see the *MLwiN* help topic "Plotting data ".
Graphic commands refer to the current 'graph set' denoted in *MLwiN* by P(N) (N=1,…10) and within each set to a specific data set, M.

15.50 Command Macro commands for updating the front end

Macro commands for updating the front end

The front end regards obeying a macro as a single action. Ordinarily, the front end is updated after each action. This can be a problem if a macro is obeying a big task, for example a simulation, where it would be helpful for the front end to be updated at regular intervals while the macro is executing. The pause command can be used to do this and in conjunction with other commands listed below the pause command can generate other useful behaviour.

15.51 Command macro editor

macro editor

From the file menu the user can access the macro editing window. This allows macros to be constructed or existing macros to be edited. Macros can be saved and a list of the previous six macros accessed is displayed. Basic editing functions are available, such as search and replace accessed from the bottom of the window.
An **execute** button instructs *MLwiN* to run the displayed macro, If an error occurs then the relevant line is displayed in red. Any line starting with 'note' is displayed in green and not executed.
If your macros call other macros without referring to the full directory (folder) path, you should ensure that the directory where they are located is specified in the directories option .

15.52 command macro example

Macro example

The following set of macros will simulate a simple 2-level variance components model with just an intercept term. The first macro is the driver macro that sets up the model, and calls two macros to generate 100 simulated data sets and fit the model to them within a loop. The macros are fully annotated.

```
Note Macro simu.txt
note create identifiers for 108 pupils
pref 0
postf 0
Generate  1 1080 1 c1
note generate identifiers for 60 schools, each with 18 students
code 60 18 1 c2
note generate constant vector in C3 C4
put 1080 1 c3
put 1080 1 c4
note name columns
name c1 'student' c2 'school' c3 'cons' c4 'resp'
note declare c4 the response variate
Response c4
note identify levels 2 and 1
iden 2 c2
iden 1 c1
note declare explanatory variables - here the intercept only
expl 1 c3
note set covariance structure at each level - here a single variance
setv 2 c3
setv 1 c3
note eras old used columns so they are blank
eras c5 c11-c15
note select bacth mode
batch 1
note set seed for simulation
seed 1
note loop for simulations
loop b1 1 100
note this file simulates 1 dataset using given parameter values
obey genresp.txt
note this file uns the model and stores results in c11-c13
obey run.txt
endloop
note summarise stored results
echo 1
aver 3 c11-c13
endobey


note macro for running IGLS  for a model
start
note join level 2 variance, level 1 variance and intercept in c11-c13
pick 1 c96 b11
pick 2 c96 b12
pick 1 c98 b13
join c11 b11 c11
join c12 b12 c12
join c13 b13 c13
endobey


note generates a simulated response
note set level 2 var. level 1 var, intercept true values
edit 1 c96 10
edit 2 c96 40
edit 1 c98 30
note now simulate from model
simu 'resp'
endobey
```

The results are as follows in the output window.

	N	Missing	Mean	s.d.
c11	100	0	10.183	1.9938
c12	100	0	39.939	1.6445
c13	100	0	29.999	0.44600

Points to note:
You can run a model within a macro file, and manipulate the results. Thus, general simulations can be carried out. Useful commands for use in macros are the following:
ERROr which allows the program to abort or not if an error occurs
CONVerge which tests whether convergence has been achieved after an iteration
IMAC in conjunction with the PAUSe command allows the user to halt execution of a macro by clicking a particular button on the toolbar.
There is also a set of commands for displaying graphics .

If you are running a model with a Normal response be careful to include commands in your macro which turn off pre and post file processing using the commands
PREFile 0
POSTfile 0

15.53 Command Macros

Macros - an introduction

A complex or frequently-used sequence of commands may be coded using a text editor and stored in an ASCII file for execution by *MLwiN* as a sequence. Such a stored sequence is called a macro. Separate macros must be stored in separate files.
Most *MLwiN* commands may be included in macros, though some will require the user's intervention during execution. The general macro commands include those for conditional execution of code, commands to control loops, and commands to ascertain the current state of a model. These commands, together with the facility for indirect addressing of boxes, columns, and groups (see General introduction: Parameter descriptors), enable the user to produce macros which are flexible and general. In addition to the general macro commands, there are commands which manipulate the graphics, commands which allow macros to create input screens commands to allow macros to specify when and how the front end is updated.
The macros are classified in the following help 'books':

- General macro commands
- Macro commands for controlling graphics
- Macro commands for updating the *MLwiN* front end
- Macro commands for controlling MCMC estimation
- Miscellaneous new macro commands

Note that any windows which are open will not be updated while macros are running, but will register changed parameter values etc. after macro execution is complete. Windows will also be updated whenever a macro is paused .
MLwiN version 2.0 incorporates a macro editor which may be accessed from the File menu.

15.54 Command Macros - configuring the macro menu

Macros - configuring the macro menu

Note that this is no longer available in *MLwiN* release 1.1 onwards.

There is now a macro editor .

15.55 Command MAVErage

MAVErage of parameters stacked in <column 1> average values to <column 2> sd's to <column 3> This command will take a set of stacked parameter values, typically arising from a simulation or bootstrap and writes the means to se's to the two output columns. The stacked column should contain repeated sets of parameters held in the order fixed parameters, followed by random parameters, highest level first. The length of the stacked parameter column must be an exact multiple of the total number of parameters(fixed + random) in the current model.

15.56 Command MDEBugging

MDEBugging <value 1>

When debugging macros it is useful to be able to interrupt a macro at any time. This can be allowed by setting the macro debugging mode to 1. When macro debugging is on a PAUSE 1 statement is automatically generated between every single macro command. Whilst useful for debugging, execution is slowed down hugely because the front end updates itself after every command in a macro is executed. Macro debugging is turned off by setting <value 1> to 0.

15.57 Command MHAStings

MHAStings

This is now superseded by the MCMC command.

15.58 Command MONItor model changes

MONItor model changes <value 1>

The front end displays converged parameters in green and unconverged parameters in blue. If a model changes, for example, explanatory variables are added or removed or the number of levels changes the model is deemed to have changed and displayed parameters will turn from green to blue. To achieve this the back end must monitor model changes and inform the front end when a model change occurs. This normally leads to desirable behaviour, however sometimes macros are programmed in such way that model settings are changed, for example temporary variables are added to the model and then removed from the model behind the scenes. These variables are really programming constructs that are required to estimate the model but do not represent parameters of interest to the user. The software can not distinguish between legitimate model changes and model changes made for programming convenience. Therefore if PRE and POST files are declared which change a model in some way, the front end will be informed after every iteration that the model has changed and the model parameters will always be displayed in blue. However, the model will stop iterating when the active set of parameters has converged. Macros can stop the back end informing the front end of model changes by issuing the command MONItor 0 which turns back end model monitoring off. Macros which make model changes as programming devices should turn monitoring off at the start of the PREFile and turn monitoring on (<value 1> = 1) at the end

of the POSTFile.

15.59 Command NFIXed

NFIXed, output number of fixed parameters to <box>

15.60 Command NLEVel

NLEVel
Output the number of levels in the current model to <box>

15.61 Command NMSTR

NMSTR c1 s1 c2 s2 etc.
Set string variable s1 to be equal to the name of C1, s2 to be equal to the name of C2 etc.

15.62 Command NRND

NRND,
output the number of random parameters {at level <value>}to <box>

15.63 Command NUNIts

NUNIts, output the number of units at level <value> to <box>

15.64 Command OBEY

OBEY the macro stored in <filename>
OBEY <*value-1*> {<*value-2*> ...} Respond to prompt with the macro <*filename*> In the second format <*value-1*>, <*value-2*>, etc., are passed as parameters to the macro in <*filename*>. They are referred to within the macro as %1, %2, etc.

15.65 Command PAUSE

PAUSE N
N=1 If this statement is in a macro the front end is updated every time this statement is executed. If a macro is running a simulation you might want to set up some graphs pointing to the columns where the simulation is storing its results. This could be done by the macro user in the front end or directly in the macros by using the graphics commands. If you place a pause 1 statement in the simulation loop then the graphs will be dynamically updated as the macro executes. If error messages are produced during macro execution, then these will be displayed and the macro halted until the **Return** key is pressed.
N=2 This has the same function as for **N=1** but does not halt macro to display any error messages. This is useful, for example, during simulations where estimation problems may arise. All error messages produced will, however, be sent to the **Output** window where they can be viewed.
PAUSE If the pause command is given with no parameters then control is returned to the front end and two buttons appear on the main tool bar. One button is labelled "Resume Macro" the other is

labelled "Abort macro". The macro remains paused (users can rearrange front end components, look at data etc.) until the resume or abort button is pushed. Once one of these buttons is pushed they both disappear and the macro is aborted or resumed accordingly.

15.66 Command POSTfile

POSTfile <filename>
Set macro file to be obeyed after STARt, NEXT, RESIduals or LIKElihood. **POST**file mode <*value*> If <*value*> = 0 disable postfile. If <*value*> = 1 enable postfile.

15.67 Command PREFile

PREFile <filename>
Set macro file to be obeyed before STARt, NEXT, RESIduals or LIKElihood. **PREF**ile mode <*value*> If <*value*> = 0 disable prefile. If <*value*> = 1 enable prefile.

15.68 Command PUPDate

PUPDate estimates to <column 1> sd's of estimates to <column 2>
This command takes a column of estimates, in the order fixed followed by random parameter estimates and a parallel column of SD's and updates the columns C1096-C1099 accordingly. The values that were in C1096 and C1098 are then treated as the previous estimates.

15.69 Command RETUrn control

RETUrn control from current macro file to caller

15.70 Command RINIt

RINIt, mode 0, show the value used for initialising random parameters when added to a model {and store in <box>}
RINIt, mode **1**, set the value used for initialising random parameters to <*value*>

15.71 Command RPARameters,

,
RPARameters
store details of random parameters in <*group*> <*group*> must consist of 3 columns. The details of each random parameter will be stored as a row of <*group*>. The first 2 columns will contain coordinates of the parameter in terms of the positions of the corresponding explanatory variables in the explanatory variable list. The third column will contain the level of the parameter. The rows are in the order of the random parameters as displayed by the RANDom command.

15.72 Command RPOSition

RPOSition at level <*value*>, of <*explanatory variable column*> in random parameter list output to <*box*>. If <*explanatory variable column*> is not in the random parameter list <*box*> is set to 0.

15.73 Command SAY<text>

SAY
Display <*text*> on screen . The sequence \n causes line feed and carriage return. For example, SAY \n\n ERROR in macro \n\n Note : the SAY command prints text whether echoing is on or off. See ECHO.

15.74 Command SEPRed

SEPRed <group> output to <column>
Takes a prediction in the same format as the PREDict command. However, instead of the predicted value being output the SD(predicted value) is output. This SD is based on the covariance matrix of the fixed parameters. To obtain SD's which incorporate sampling variability of residual estimates, the RFUN command should be used.

15.75 Command SJOIn

SJOIn <string 1> "TEXT" <string 2>
This commands concatenates string literals and variables. For example SJOIN "This is " S1 SJOIN S1 "some text " S2 would result in S2 containing the string "This is some text".

15.76 Command SPMC

SPMC <value 1>
This command is now redundant.

15.77 Command string button

WBUTton for string
WBUTton S Creates a button with text string S printed on it. The following set of commands create a window that contains a button labled do work, that calls a the macro work.txt when the button is pressed. windex 0 0 wdesc "" "c:\work.txt" sjoin "do work" s1 wbut s1 wset 0 1

15.78 Command STRIngs

STRIngs displays contents of the 20 string variables.

15.79 Command SUPPress arithmetic warnings

SUPPress arithmetic warnings

15.80 Command SWITch,

SWITch **CASE**, **LEAV**e, **ENDS**witch are components of a conditional sequence. For

example
SWITch B1 CASE 1 : statements to be obeyed if B1 = 1
LEAVE CASE 2,3,4 : statements to be obeyed if B1 = 2, 3 or 4
LEAVE CASE : statements to be obeyed if neither of the above cases is true
ENDSwitch Note that commands should not be typed on the same line as the CASE command.

15.81 command TLRA

TLRA C1 C2 C3

Suppose we have a set of cut points defined for a logistic distribution in C1. If the typical value is X and if the corresponding value in C2 = 0 then a random draw from the truncated standard logistic* distribution in the range (-∞∞, X) is selected and placed into the corresponding position in C3. If the corresponding value in C2 = 1 then a random draw from the truncated standard logistic distribution in the range (X, ∞∞) is selected and placed into the corresponding position in C3.
*The density function for the standard logistic distribution is given by

$$f(x) = \frac{\exp(x)}{[1+\exp(x)]^2}$$

which has mean 0 and variance 3.29

15.82 command TNRA

TNRA C1 C2 C3

Suppose we have a set of cut points defined for a Normal distribution in C1. If the typical value is X and if the corresponding value in C2 = 0 then a random draw from the truncated standard Normal distribution in the range (-∞∞, X) is selected and placed into the corresponding position in C3. If the corresponding value in C2 = 1 then a random draw from the truncated standard Normal distribution in the range (X, ∞∞) is selected and placed into the corresponding position in C3.

15.83 Command GCLR

Graph colour

GCLR N Set colour for current data set of current graph.. The 16 colour numbers are given on the customised graphs, plot style, colours drop down list.

15.84 Command WMSG

WMSG "message text"

This command will cause the given message to be displayed in a windows message box when control next returns to the front end by a PAUSE or ABORT command or the macro terminating normally.

15.85 Command YVAR

YVAR output column number of response variable to <box>

mcmc commands in macros

16.1 Command XCLA

XCLA <value>
This command toggles off (value = 0) and on (value = 1) whether the current model is cross-classified or nested

16.2 Command BDIC

BDIC <b1> <b2> <b3> <b4>
This commands outputs the DIC model comparison criterion along with the effective number of parameters in the model. The optional arguments will output to boxes the following: <b1> DIC, <b2> pD – effective number of parameters, <b3> mean deviance Dbar, <b4> deviance at mean parameter values D(thetabar).

16.3 Command BUGI

BUGI <fname1> <fname2> <c1>
This command takes as input the two files produced by WinBUGS with extensions .out for fname1 and .ind for fname2. It also requires a first column <c1> and will then input the variables from WinBUGS to consecutive columns starting from <c1>.

16.4 Command BUGO

BUGO <value 1> <value 2> <value 3> {residual starting values in C1}{informative prior definition in C2} output filename <fname>.

This command will allow the user to output the currently set up model into WinBUGS code in filename <fname>. It requires three other arguments: value 1 gives the version of WinBUGS where 3 = version 1.3 and 4 = version 1.4. The second argument, value2 is the response type that, as with the MCMC command, takes values 1 for Normal, 2 for Binomial, 3 for Poisson and 4 for multivariate normal. The third argument value 3 is the link function and takes values 0 for logit, 1 for probit, 2 for complementary log-log and 3 for log link. For normal and multivariate normal models this is set to 0. The user may optionally set the residual starting values (C1) and informative prior definition (C2). For these arguments the format is the same as given in the MCMC command.

16.5 Command DAFA

DAFA Estimates to <c1> standard errors to <c2>.

This command outputs the estimated factor values (and standard errors) from a factor analysis model to a column. Note for models with several factors these will be stacked in the order they are defined in the factor window/command. Also note that if a factor is defined at a level higher than the lowest level of the data, then its estimates will be augmented with zeroes to make it the same length as the lowest level.

16.6 Command DAFL

DAFL Estimates to <c1> standard errors to <c2>.

This command outputs the estimated factor loadings (and standard errors) from a factor analysis model to a column. Note for models with several factors these will be stacked in the order they are defined in the factor window/command

16.7 Command DAFV

DAFV Estimates to <c1> standard errors to <c2>.

This command outputs the estimated factor variance matrix (and standard errors) from a factor analysis model to a column. Note for models with several factors this will be stored as the below diagonal portion of the matrix with zeroes where factors are uncorrelated.

16.8 Command DAMI

DAMI

DAMI 0 current iteration estimates to <c1>
DAMI 1 estimates to <c1>
DAMI 2 estimates to <c1> standard errors to <c2>
This command outputs a complete (i.e. including non missing responses) response variable, y to column <c1>. If the mode is 0 then this response variable will be the value of y at the current iteration, which can be used for multiple imputation. If the mode is 1 the value of y will be the mean estimate from the iterations produced. Mode 2 is as for mode 1 but with additionally the standard errors of the estimate stored in column <c2>.

16.9 Command Fact

Fact

This command will set up the factors for a factor analysis model and has many arguments. The format is as follows
FACT <value 1> <value 2> <value 3> ((<value 4> << <value 5> <value 6> >> <value 7> <value 8>)) [[<value 9> <value 10> <value 11> <value 12>]]
Where << >> is repeated <value 1> times, (( )) is repeated <value 2> times [[]] is repeated

<value 3> times.
<value 1> is the number of responses, <value 2> is the number of factors, <value 3 > is the number of correlated factors; For each factor <value 4> is the level/classification for the random part of the factor. For each loading, <value 5> is the starting value for the factor loading, and <value 6> is an indicator of whether the factor loading is constrained (1) or not (0). For each factor, <value 7> is the starting value of the factor variance and <value 8> is an indicator of whether the factor variance is constrained (1) or not (0). For the correlated factors, <value 9> is the first factor number, <value 10> is the second factor number, <value 11> is the starting value for the covariance and <value 11> indicates if this covariance is constrained or not.

16.10 Command LCLO

LCLO <value>
This command toggles on/off the possible forms of complex level 1 variation when using MCMC. By default (value=0) we express the level 1 variation as a function of predictors. If this is toggled (value=1) we express the log of the level 1 precision (1/variance) as a function of predictors

16.11 Command MCCO

MCCO <value>
This command allows the user to have constrained settings for the lowest level variance matrix in a multivariate Normal model. The default value for value, 0, is used for a full covariance matrix. Four other settings are currently available: 1 - all correlations equal and all variances equal; 2 – an AR1 structure with all variances equal; 3 – all correlations equal but independent variances; 4 – an AR1 structure with independent variances.

16.12 Command MCMC

MCMC
This command performs MCMC estimation for the currently set up model. It is advisable to run an IGLS or RIGLS run before running the MCMC methods. There is a different syntax for the burn-in and the main run of the simulation. A burn-in (of length 0 if necessary) MUST be run first to set up the model.

The syntax for the burn-in is as follows:

MCMC <0> burn in for <value 1> iterations, use adaptation method <value 2>, scale factor <value 3>, %acceptance <value 4>, tolerance <value 5> {residual starting values in C1, s.e. residual starting values) in C2} {priors in C3} fixed effect method <value 6> residual method <value 7> level 1 variance method <value 8> other levels variance method <value 9> default prior <value 10> model type <value 11>.

This command starts the MCMC sampler and burns in for <value 1> iterations. The settings

for <value 2> to <value 5> will only come into effect if some parameters are updated by MH sampling but must be given.

Value 1 = no of burn in iterations.
Value 2 = 1 if adaptation is to be used, 0 otherwise.
Value 3,Value 4 and Value 5 are explained in the help system (See under MH Settings). Value 4 and Value 5 are ignored if Value 2 = 0).

The user has the option of supplying starting values for residuals at level 2 and above. These should be stacked into 1 column C1, higher level residuals first. If you have more than one set of residuals at a level then residuals corresponding to explanatory variables with lower number get stacked first. If you choose to supply residual starting values then you must stack residuals for every random variable at level 2 and above into the input column.
C2 is required if C1 is specified. This column should contain the standard error matrices for the residuals. These should be in the same format as the standard errors output from the RESI command when the RCOV command takes value 2 i.e. if there is more than one parameter random at a level then the complete (lower triangle) covariance matrix for the residuals at that level is required. The stacking should be in the same level order as for C1.

The user also has the option is to supply prior information (C3) for the parameter estimates. The components for which prior information can be specified differ in the fixed and random parts of the model. In the fixed part prior information can be supplied on a per parameter basis. In the random part of the model prior information is displayed on a per level basis, that is prior information must be given for the full covariance matrix at a given level. For fixed parameters prior information takes the form of a mean and SD, for covariance matrices prior information takes the form of estimate values and a sample size. See the section on "Priors" in the MLwiN help system for more details on the meaning of these terms. The format that the prior column should take is described in the PRIOr command.

As the MCMC command is generic <value 6>, <value 7>, <value 8> and <value 9> need to be input for the sampler to know which methods to use for each set of parameters. These parameters take the values 1 for Gibbs Sampling, 2 for univariate MH Sampling and 3 for multivariate MH Sampling. Note that not all combinations of methods are available for all sets of parameters and all models.

<value 10> indicates which default priors are being used for variance parameters. This parameter takes the value 1 for Gamma priors or 0 for Uniform on the variance scale priors. See the section on "Priors" in the MLwiN help system for more details on the meaning of these default priors.
<value 11> indicates the model type being fitted. This parameter takes the values 1 for Normal models, 2 for Binomial models, 3 for Poisson models and 4 for multivariate Normal models.

The syntax for the main chain is as follows:

MCMC <1> continue for <value 1> (stored) iterations, thinning every <value 2> iterations, appending stacked trace to C1, likelihood to C2, mean of all updates to C3, standard deviation of all updates to C4, run <value 3> Metropolis Hastings updates per iteration for model type <value 4>.

The MCMC 1 command continues with the settings used in the last MCMC 0 command. The sampler will run for <value 1>*<value 2> iterations but only store every <value 2>th iteration to store a total of <value 1> iterations. These <value 1> iterations are stored in C1, an iteration at a time in the order fixed parameters followed by random parameters, highest level first. The likelihood column holds the likelihood value for each stored update. The final two columns are the means and standard deviations of parameters taken from all updates irrespective of the thinning setting. For each iteration any steps run by MH sampling will be updated <value 3> times, whilst any Gibbs steps are run just once. Model type <value 4> is identical to the MCMC 0 command.

16.13 Command MCRE

MCREsiduals
This command calculates residuals from an MCMC model. The residuals setting screen must be filled out in the same way as you would for IGLS/RIGLS models. Typing MCRE will then calculate residuals based on the current MCMC model run. Note that the residual level to be output (RLEV), the type of variances output (RCOV) and the other options (RTYP, RFUN, ROUT) are set by their particular commands before MCRE is called.

16.14 Command MCRS

MCRS <value>
This command sets the MCMC random number seed to integer <value>. Note that the MCMC procedures use a different seed from the rest of MLwiN which uses a seed that can be set using the **SEED** command. The random numbers generated within the MCMC procedures use the Wichmann-Hill generator for uniform random numbers that actually has 3 seeds and the **MCRS** sets the first of these seeds (the other two are set to 4 and 1972 by the **MCRS** command).

16.15 Command MERR

MERR <N> {value 1a value 1b …. value Na value Nb}
This command sets up measurement errors on predictor variables. Here N defines the number of variables that have measurement errors. Then for each variable with measurement error a pair of inputs is required: value Ma is the explanatory variable number for the M^{th} variable which has measurement error and value Mb is the variance of the measurement errors for the M^{th} variable

16.16 Command MULM

MULM <L1> <N1> {<C1> <C2>}
This defines a multiple membership across N1 units at level L1. IF N1>1 i.e. if there is multiple membership then the column number C1, which is the length of the dataset, will contain the

first set of weights for the multiple membership. Note that there should be N1 weight columns and they should be sequential in the worksheet starting from C1. If the response is multivariate then the column number C2 must be input and this contains the first set of identifiers for the classification. Note that for a p-variate model each lowest level unit contains p records and the identifiers (sequence numbers) for each response variate need to be extracted into C2 and following columns. There should be N1 of these identifier columns and they should be sequential starting from C2 in the multivariate case.

16.17 Command PRIOr

PRIOr <C>

Informative priors are defined in column C. Note that this command is used in the interface but is redundant when writing macros as the MCMC command has the optional argument that actually sets up the informative prior.

The prior column (C) takes information for the fixed effect parameters followed by the random parameters, highest level first. Before each component there is an indicator 0 for no prior information and 1 if there is prior information. If the indicator is 1 the prior information follows. For a 2 level random coefficient model, with just an intercept and slope in the fixed part, if we wanted to specify the following prior information

Fixed intercept : prior (10,3)
Fixed slope : no prior information
Level 2 random part : intercept variance (5), slope variance (0.5), covariance (1), sample size 50
Level 1 random part : no prior information

We should put the following numbers in the prior column:
1 10 3 0 1 5 1 0.5 50 0

16.18 Command PUPN

PUPN <C1> <C2>

Current MCMC (mean) estimates in <C1> variance matrix of estimates in <C2>. This command takes a column of estimates in the order fixed followed by random parameter estimates and a second column containing the (stacked) variance matrix of the fixed parameters followed by the (stacked) variance matrix for the random parameters and updates the columns c96-c99 accordingly. The values that were in c96 and c98 are then treated as the previous estimates. Note that this command replaces the command PUPD as it now also updates the parameter covariances to allow multiple tests and confidence intervals.

16.19 Command STKRank

STKRank chain in C num iters N, centiles N.N output columns C..C

The stkrank command takes a chain of *t* iterations for *m* parameters in a column ordered:

p_{00}, p_{10}, p_{20},p_{30}..p_{m0},p_{01},p_{11}..p_{m1}….p_{t0},p_{t1}..p_{mt}

and calculates specified centiles for the rank of each parameter.

This command is particularly useful for calculating interval estimates for the ranks of residuals. Note that to store residuals for an MCMC run select <model><mcmc><store resuduals> from the menu structure.

Note that with a random slopes model at level 2 and MCMC storage of level 2 residuals specified, then MCMC outputs the residuals to a single column in the following format

U01[0],U11[0]….U0J[0],U1J[0]…….U0J[T],U1J[T]

Where J = number of level 2 units and T = number of iterations. In this case the intercepts and slopes need to be split into two separate columns for use with the STKRank commands. That is, assuming combined residuals are output to c301 and 5000 iterations

```
CODE 2 1 5000 c300
SPLI c301 c300 c302 c303
STKRank c302 5000 5 50 95 c311-c313      [centiles of ranks of intercept residuals]
STKRank c303 5000 5 50 95 c314-c316      [centiles of ranks of slope residuals]
```

The mcmc store residuals function produces a chain of residuals that does not include missing values for any missing units. Units can be missing if all the data rows for that unit contain one or more items of missing data. When we derive rank information from the residual chain it may be useful to insert missing values, in the output columns from the STKRank command corresponding to any missing units. For example, if we have plotted out the ranks and their confidence intervals, we may want to identify units by clicking in them. If the plotted data set is of the shorter length(excluding missing units) the identify point routines will not form a match with a classification. This is because a point is identified as belonging to a classification if

(length of plotted data set containing point = number of units in a classification(including missing units).

The output from the STKRank command can be realigned to contain missing units by the REALign command :

16.20 Command TIMEr

TIMEr <value> {<B1>}

This computes the time for an MCMC run, when used in a macro.
Value = 1 Start the timer to time a set of commands
Value = 0 Record the time since timer started and display in output window and (optionally) store in box B1.

16.21 Command WRAN

WRAN <N1> <N2> <C1> <C2…..CN>

Generates <N1> draws from a Wishart distribution with parameters, degrees of freedom <N2> covariance matrix (S) stacked in lower triangle form by columns in <C1> where the number of elements of S is N-1. The <N1> draws are output to the columns <C2…CN> with one element of the matrix in each column (again in lower triangle form).

16.22 Introduction to mcmc commands

Introduction to MCMC commands

The macro commands that are associated with the MCMC options in MLwiN version 2.0 can be split into 4 groups; commands that set up an MCMC model, commands that run the MCMC sampler, commands that send output to columns, and commands linked with the WinBUGS interface.
For an introduction to MCMC sampling and detailed examples in MLwiN see Browne (2002). This volume can also be downloaded from the MLwiN web site.

MCMC model setup commands
FACT sets up a factor analysis model
LCLO controls complex level 1 estimation
MCCO allows the user to choose certain patterned, lowest level, covariance structures
MCRS sets the random number seed for MCMC procedures
MERR sets up measurement error estimation
MULM defines a multiple membership model setup
PRIOr sets up informative priors
XCLA specifies whether model is cross-classified or nested.

Model running commands
MCMC sets up options for burn in and main chain run
PUPN updates (default) columns (c1096-c1099) where fixed and random parameter estimates are held
TIMEr calculates the time for an MCMC chain
WRAN generates draws from a Wishart distribution

Model output commands
BDIC Outputs the DIC (model comparison) statistic
DAFA Outputs factor values
DAFL Outputs factor loadings

DAFV Outputs factor covariance matrices
DAMI Outputs current iteration estimates for responses (including imputed)
MCRE calculates residuals

WinBUGS interface commands
BUGI takes WinBUGS output files for input to MLwiN
BUGO produces output files for estimation in WinBUGS.

Miscellaneous commands

17.1 Command ECHO

ECHO {<value>}
If <*value*> = 0 turn echoing off. If <*value*> = 1 turn echoing on (the default setting). If <*value*> is absent, alternate between on and off. For example, if echoing is on, input data are displayed on the screen as they are read in; and commands coded in macros are displayed, with their results, as they are executed. Turning echoing off suppresses such display.

17.2 Command LOGAppend

LOGAppend <filename>
Resume logging of commands and displayed output, appending to the contents of <*filename*> which must be an existing file.

17.3 Command LOGO

LOGOn or off <filename> {according to <value>}
Use this command to control whether or not *MLwiN* keeps a 'log' of displayed commands and output. Such a log is stored as an ASCII file. If <*value*> = 0 turn logging off (the startup state). If <*value*> = 1 turn logging on. If <*value*> is absent, alternate between on and off.

17.4 Command NOTE

NOTE <text>
Insert the remark <*text*>. Useful for documenting logged output. Otherwise ignored.

17.5 Command WAIT

WAIT {<value>}
From version 1.03 onwards. This command is now obsolete.

Matrix operation commands

18.1 Command IMATrix

IMATrix, output the identity matrix of rank <value> to <output column>

18.2 Command: Matrices using the calc command

The following matrix operators and functions are available within a CALC command when at least one of the associated operands is a matrix:

+	add outer
-	subtract outer
*	multiply outer
/	divide outer
^	raise to power outer
~	transpose of
diag	diagonal of
det	determinant of
inv	inverse of
sym	symmetric matrix of
hsym	half symmetric matrix of

18.3 Command Matrix operations

```
          C1           C2           C3
 N =       4            4            4
   1  1.0000       2.0000       3.0000
   2  4.0000       5.0000       6.0000
   3  7.0000       8.0000       9.0000
   4  10.000       11.000       12.000
prin g2
          C396         C395         C394
 N =       4            4            4
   1  2.0000       4.0000       6.0000
   2  8.0000       10.000       12.000
   3  14.000       16.000       18.000
   4  20.000       22.000       24.000
calc g3=g1*g1
prin g3
          C393         C392         C391
 N =       4            4            4
   1  1.0000       4.0000       9.0000
   2  16.000       25.000       36.000
   3  49.000       64.000       81.000
   4  100.00       121.00       144.00
```

Outer operations are defined for matrices and vectors provided the length of the vector equals the number of rows in the matrix.

```
inpu c1
1 2 3 4
calc g3=g1*c1
prin g3
        C393         C392         C391
N =       4            4            4
  1  1.0000       2.0000       3.0000
  2  8.0000       10.000       12.000
  3  21.000       24.000       27.000
  4  40.000       44.000       48.000
```

Operations on columns as matrices

Using groups for matrices is convenient when we have small numbers of small matrices. When this is not the case we can soon run out of *MLwiN* columns to store the matrices. To get round this problem it is possible to assign matrix dimensions to an *MLwiN* column. If this is done the column behaves like a matrix in the CALC command . The commands MATR and MDIM are available:

It is important to distinguish between the two meanings of the word 'column'. An *MLwiN* column, such as *<input column>*, is one of 400 such storage locations in *MLwiN* (400 is the default). Typically such a column does not itself have matrix dimensions, but if it has been assigned dimensions *r* by *c* then its *c* 'matrix columns' correspond to successive sequences of *r* data items within the storage space. This allows matrix operations to be performed on the data in the *MLwiN* column. It remains possible also to treat it as a single column if necessary. For example,

```
gene 12 c1
matr c1 4 3           [treat c1 as a 4 by 3 matrix]
calc g5=c1
prin c1
          C1
 N =      12
   1  1.0000
   2  2.0000
   3  3.0000
   4  4.0000
   5  5.0000
```

```
   6  6.0000
   7  7.0000
   8  8.0000
   9  9.0000
  10  10.000
  11  11.000
  12  12.000
prin g5
        C400         C399         C398
 N =      4            4            4
   1  1.0000       5.0000       9.0000
   2  2.0000       6.0000       10.000
   3  3.0000       7.0000       11.000
   4  4.0000       8.0000       12.000
```

If an *MLwiN* column is assigned matrix dimensions *r* by *c* the first *r* entries in the *MLwiN* column are taken as the first column in the matrix, the next *r* entries as the second column in the matrix and so on. We can freely mix matrix columns and groups in the CALC command. For example,

```
calc g6=g5*.(~c1)+(c1+3)*.(~c1)
prin g6
        C397         C396         C395         C394
 N =      4            4            4            4
   1  259.00       298.00       337.00       376.00
   2  289.00       334.00       379.00       424.00
   3  319.00       370.00       421.00       472.00
   4  349.00       406.00       463.00       520.00
```

The command MDIM shows any dimensional information associated with a column. For example,

```
mdim c1
4 rows by 3 columns
```

If a column is assigned a matrix result from the CALC command the appropriate dimensions for that column are set :

```
calc c4=g5*.(~c1)+(c1+3)*.(~c1)
mdim c4
4 rows by 4 columns
```

Square matrix inversion also is supported, as the example below, which simulates a data set and does an OLS fit, shows.

```
uran 300 c1           [generate x's]
matr c1 100 3         [dimension x's as a 100x3 data matrix]
gene 3 c2             [choose β=(1,2,3)]
nran 100 c3           [pick normal errors]
calc c4=c1*.c2+c3     [form y = xβ + e]
calc g1=inv(~c1*.c1)*.(~c1*.c4)    [calculate β = (x^T x)^-1 x^T y]
prin g1                            [print β]
        C393
 N =      3
   1  0.94252
   2  2.0032
   3  3.1838
```

We can also calculate the determinant of a matrix :

```
calc b1=det(~c1*.c1)
   5383.7
```

Sometimes *MLwiN* stores only half of a symmetric matrix, for example the covariance matrices in C97 (for random parameters) and C99 (for fixed parameters),and covariance matrices output from the RESI command. The matrix operators of the CALC command operate only on complete matrices. A symmetric matrix with half of its elements omitted can

be turned into a complete matrix using the SYM function in the CALC command. For example:

```
inpu c1
1
2 3
4 5 6
7 8 9 10
calc g1=sym(c1)
prin g1
        C392          C391          C390          C389
 N =       4             4             4             4
   1  1.0000        2.0000        4.0000        7.0000
   2  2.0000        3.0000        5.0000        8.0000
   3  4.0000        5.0000        6.0000        9.0000
   4  7.0000        8.0000        9.0000        10.000
```

We can return a symmetric matrix to its triangular storage form using the HSYM command. For example :

```
calc c2=hsym(g1)
```

The diagonal of a matrix can be retrieved :

```
calc c2=diag(g1)
prin c2
        C2
 N =       4
   1  1.0000
   2  3.0000
   3  6.0000
   4  10.000
```

18.4 Command MATRix

MATRix, declare data in <input column> to be a matrix with <value> rows and <value> columns

18.5 Command MDIMension

MDIMension, dimensions of matrix in <input column> {number of rows to <box>, number of columns to <box>}
If boxes are not specified dimensions are displayed only.

18.6 Command Principal components

PCOM principal components
PCOM on <N> variates stored in C C C… {store eigenvectors in C C C …}
The output appears in output window

Multivariate-multinomial commands

19.1 Command MNOM

MNOM mode M short response in C long response output to C, indicator variable to C, reference category [N]
Sets up a multinomial response vector into 'long response output column'. See multicategory

response example for details.

Mode 0 –unordered
Mode 1 ordered

Example:

```
MNOM 1 'a-point' c10 c11 6
Name c10 'resp' c11 'respcat'
Iden 1 c11 2 'pupil' 3 'estab'
Addt 'cons'
Rpat 1 1 1 1 1
Addt  'cons'
Setv 2 'cons.12345'
```

19.2 Command MVAR

MVAR 1

Turn multivariate mode on,
warning: if a univariate model is setup, it will be cleared.

MVAR 1 C C

Add C C as reponses in a multivariate model. The responses are stacked into a column called "resp" and another variable called "resp_indicator" is created and declared as the level 1 id.

MVAR 0 C C

Remove responses C C from a multivariate model.

Example:

We can set up a multivariate model with three responses by

```
MVAR 1
MVAR 1 "r1" "r2" "r3"
ADDT 'CONS'
IDEN 2  'student'
SETV 2
```

19.3 Command RDIST

RDIST response (in order) number N, type M
Specifies response type (M) for response number (N)

M = 0 binomial, 1, poisson, 2 negbinom, 3 normal, 4 multinomial, 5 ordered multinomial

19.4 Command RPAT

RPAT N N…

In a multinomial or multivariate model terms can have common coefficients across several responses or response categories. The pattern of commonality is specified with the RPAT command. Thus, in a multivariate model with 4 responses :

RPAT 1 1 1 0

Specifies that any terms subsequently added with the ADDTerm command have common coefficients across responses 1..3.

RPAT

With no parameters specifies that separate coefficients are to be added for each response or response category by subsequent ADDTerm commands.

Simulation commands

20.1 Command BOOTstrap

BOOTstrap

BOOTstrap sample <*value*> records from <*input group*>, results to <*output group*> For example, if C1 = (1 2 3 4), BOOT 1000 C1 C2 samples with replacement 1000 values (rows) from C1 and outputs the results to C2. *Note that the only situation where the non-parametric bootstrap is always known to provide consistent estimators for a multilevel model is for resampling of the highest level units. A paper on this issue can be downloaded from the* mlwin web site.

20.2 Command BRANdom

BRANdom

BRANdom <*value*> random numbers to <*output column*>, from the Binomial distribution with probability <*value*> | <*probability column*> and number of trials <*value*> | <*trials column*> If neither <*probability column*> nor <*trials column*> is specified, the random numbers are generated from a Binomial distribution with a fixed probability and number of trials defined by the second and third <*value*> parameters. If either <*probability column*> or <*trials column*> is specified, it must contain a value for each random number to be generated. Random numbers are then generated using these values in sequence as the

appropriate parameter for the Binomial distribution. It is possible to specify, for example, variable probabilities and a fixed number of trials.

20.3 Command CRANdom

CRANdom

CRANdom <*value*> random numbers to <*output column*>, from the Chi squared distribution with <*value*> | <*df column*> degrees of freedom If <*df column*> is specified it must contain a value for each random number to be generated. Random numbers are then generated using these values in sequence as the appropriate parameter for the Chi squared distribution.

20.4 Command DRANdom

DRANdom

DRANdom <*value*> random numbers to <*output column*> using the frequency distribution specified in <*frequency column*> The first value in <*frequency column*> specifies the frequency of the value 0, the second the frequency of the value 1 etc. For example if C2=(1 4 5) DRAN 500 C1 C2 will generate 500 numbers in C2, containing on average 10% zeros, 40% 1s and 50% 2s.

20.5 Command ERANdom

ERANdom

ERANdom <*value*> random numbers to <*output column*> from the Exponential distribution with mean <*value*> | <*mean column*> If <*mean column*> is specified it must contain a value for each random number to be generated. Random numbers are then generated using these values in sequence as the appropriate parameter for the Exponential distribution.

20.6 Command GRANdom

GRANdom

GRANdom <*value*> random numbers to <*output column*> from the Gamma distribution with shape parameter <*value*> | <*shape parameter column*> and unit scale parameter. If <*shape parameter column*> is specified it must contain a value for each random number to be generated. Random numbers are then generated using these values in sequence as the shape parameter for the Gamma distribution.

20.7 Command HRANdom

HRANdom

HRANdom generate a hierarchy with unit numbers at each level taken from Normal distributions with means in <*column-1*> and standard deviations in <*column-2*>, storing unit identifiers in <*output group*> Each of <*column-1*> and <*column-2*> must contain a number of values equal to the number of levels in the hierarchy. The first row of this pair of columns defines the sampling distribution for the number of units at the highest level. The second row defines the sampling distribution for the number of units at the next level down which are contained in each highest-level unit, and so on, down to the final row which defines the sampling distribution for the number of level-1 units in each level-2 unit. <*output group*> must contain one column for each level in the hierarchy. Unit identifiers for each level will be stored in successive columns of <*output group*>, starting with the highest level. If you wish

to specify a definite number of units at a given level, set the corresponding element in <*column-2*> to 0. A balanced hierarchy is generated if all values in <*column-2*> are set to zero. For example, if C1 = (100, 5, 30), C2 = (0, 0, 0), and C3 = (0, 1, 4), then HRANdom C1 C2 C11-C13 will generate in C11-C13 the unit identifiers for a 3-level hierarchy with 100 level-3 units, each containing 5 level-2 units, each containing 30 level-1 units. HRANdom C1 C3 C11-C13 will generate a 3-level hierarchy with 100 level-3 units, but the number of level-2 units in each level-3 unit will have a Normal distribution with mean 5 and standard deviation 1. Similarly, the number of level-1 units in each level-2 unit will have mean 30 and standard deviation 4.

20.8 Command MRANdom

MRANdom

MRANdom a set of multivariate random Normal variates, each occupying a column <*value*> long, with mean zero and covariance matrix in <*covariance matrix column*>, the variates to be stored in <*output group*> <*covariance matrix column*> must contain the lower triangle of the covariance matrix, in stacked row order. Means other than zero can be produced by subsequent use of the CALC command. MRANdom a set of random Normal variates to be added to the existing values in <input group> with mean zero and covariance matrix in <covariance matrix column> the variates to be stored in <output group>. The number of variate sets is equal to the length of *input group.*

20.9 Command NRANdom

NRANdom

NRANdom <*value*> random numbers to <*output column*> from the standard Normal distribution

20.10 Command PRANdom

PRANdom

PRANdom <*value*> random numbers to <*output column*> from the Poisson distribution with mean <*value*> | <*mean column*> If <*mean column*> is specified it must contain a value for each random number to be generated. Random numbers are then generated using these values in sequence as the appropriate parameter for the Poisson distribution

20.11 Command SEED,

SEED

SEED re-seed the random number generator using the start value <*value*>

20.12 Command SIMUlate

SIMUlate to response column

Use the current model estimates to generate a new set of response variable values with the same structure, assuming multivariate Normality of residuals, and place these values in <*response column*>. For example, SIMU C100 will use the current fixed parameter estimates to generate predicted response values for the fixed part of the model. Then, at each level, the random parameter estimates are used to produce a set of residuals by random number generation. These residuals are multiplied by the appropriate design vector and added to the response values predicted from the fixed part and the results placed in C100.

An alternative form of the command useful when residuals are not all multivariate Normal, for example in Binomial models, is: **SIMU**late {<*value-1*> {<*value-2*> ...}} <*random part column*> In this form, that is with at least one <*value*> parameter, the command simulates the random part only of the model. <*value-1*>, <*value-2*>, etc., specify the level(s) for which residuals are simulated before being multiplied by the appropriate design matrix to form the random part at each level. The random parts for each level are then summed and output to <*random part column*>.

20.13 Command URANdom

URANdom

URANdom <*value*> random numbers to <*output column*> from the Uniform distribution on the interval (0,1)

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -