The *MLwiN* Command Interface

By

Jon Rasbash
Harvey Goldstein
William Browne
Min Yang
Geoff Woodhouse

Version 1.3

November 2000

ISBN: 085473 6115

Contents

1 An introduction to the MLwiN command interface

With release 1.1 nearly all of the original MLn commands have been 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. The following description is also available in the *MLwiN* help system.

1.1 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. A full description of *MLwiN* including installation is given in the Users' guide (Rasbash et al., 2000).

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. 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 (Rasbash et al., 1999).

Worksheets saved under *MLn* are recognised by *MLwiN* and can be retrieved. You may also save worksheets in MLn format.

1.2 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. Note that groups do not exist as independent entities. Thus, for example, erasing a group will also erase the columns linked to it.

MLwiN provides 400 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. 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
```

Columns C96, C97, C98, and C99 are reserved. *MLwiN* uses these columns to store the latest estimates of the parameters of the current model, and their covariance matrices, as follows:

C96	random parameter estimates
C97	covariance matrix of random parameter estimates
C98	fixed parameter estimates
C99	covariance matrix of fixed parameter estimates

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 www.ioe.ac.uk/mlwin/.

In release 1.0 and later of *MLwiN* the macros for fitting multilevel generalised linear models are handled via a text editing interface accessed from the **File** menu. Further macros and instructions for their use are provided in a further document '*MLwiN* macros for advanced multilevel modelling' (Yang et al., 2000). These macros will fit models for multicategorical ordered and unordered responses and for time series in discrete or continuous time.

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.

1.4.1 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.

1.5 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.

1.6 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:*MLwiN*\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.

1.7 Commands for use in version 1.1

Certain commands in version 1.1 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
CLEAr	Clears all current model specifications
DISCard	Removes (numbered) rows from the stored data matrix
ERASe	Erases specified columns
FCONstraints	Linear constraints on fixed parameters
FSDE	Standard error type (robust, model based) for fixed parameters
LINK	Assigns columns to a group
LOGAppend	Appends output logging to existing file
LOGOn	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
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
SIMUlate	Simulates from a specified Normal model to an output column
SUBSymmetric	Forms random parameter design matrices for blocks using sums
SURVival	Generates a vector for input to Cox survival models – see advanced macros (Yang et al., 2000)

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

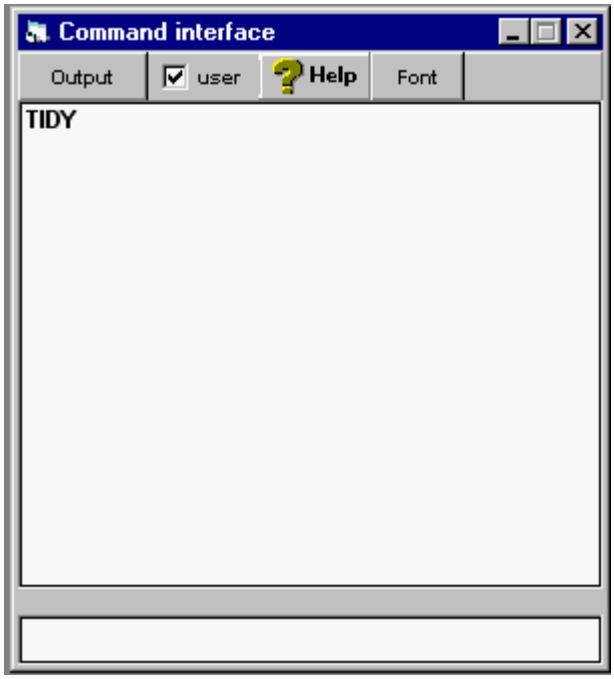
There is also a set of commands to manipulate the various matrices used by the IGLS algorithm (see later section).

2 Getting started with commands

The commands remaining in version 1.1 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 section before proceeding.

Commands are entered into *MLwiN* via a command interface window, accessed from the **data manipulation** menu, as shown below.



The above command storage box contains a single command (to tidy the worksheet) which is one of the standard commands a user may wish to access. Only the first 4 characters of a commend 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 ↵ 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.

The output window is opened by default and here 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

The following two tables list some of the standard and advanced commands you may wish to use, together with brief descriptions of their syntax.

Standard Commands	
DISCard	Eliminates a set of defined rows from a collection of columns.
EXCLude	Excludes specified cases from analysis
FCON	Constrains a linear function of the fixed coefficients.
FSDE	Sets the type of standard errors to be used for fixed coefficients (e.g. model based or sandwich)
LGRId	Evaluates the (Normal) likelihood over a grid of parameter values.
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.
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
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.
PICK	Picks a set of defined rows from a collection of columns.
RCON	Constrains a linear function of the random parameters

	parameters.
REFLate	Reflates residuals - used for nonparametric bootstrap
RSDE	Sets the type of standard errors to be used for random parameters (e.g. model based or sandwich)
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.
TIDY	Tidies the worksheet to allow more efficient use of space.

Advanced Commands	
Cross-classified models:	
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 version 1.1*
Multiple membership models:	
ADDM	Adds indicator columns
WTCOl	Assigns weights
Matrix operations	Defining and manipulating matrices - see relevant section
Access to multilevel algorithm	Provides access to the computational components of the IGLS algorithm - see relevant section.
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
SUBSymmetric	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*.
SETD	Sets up a design vector specified by the user for a random parameter.
SEED	Sets a seed for random number generator.
BOOT	Samples with replacement from a column.

3 Worksheet management

CATN mode N C 1 S 2 S … assign or clear category names to data in a column

Where S is a string

For example

CATN 1 C3 0 “boy” 1 “girl”

If a string is identical to a column name then ambiguity can be avoided by placing the \ at the start of the string. For example, if we already have columns called low, mid and high

CATN 1 c5 0 ’\low’ 1 ‘\mid’ 2 ‘\high’

Will create category names low, mid and high attached to codes 0,1,2 in column 5. The \ simply instructs MLwiN’s parser to treat the string as text and not a column name.

CATN 0 C

Clear all category information for column C

CTOG C G

create group containing column numbers listed in C

ERASe <*group*>

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

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 subsequent use in a macro then

LINK 5 G1

FILL G1

LINK 5 G2

Will prevent G2 from containing the same columns as G1.

INITialise {<*value-1*> {<*value-2*> {<*value-3*> {<*value-4*> {<*value-5*>}}}}}

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).

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 -(N+10) to <*G*>.

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

(with no parameters)

Display column numbers of all marked columns.

MARK 0, do not issue overwrite warnings for marked columns

MARK 1, issue overwrite warnings for marked columns

MOVE columns as necessary to close gaps of empty columns in the sequence.

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.

NAME <*C*> <*name*> {<*C*> <*name*> ...}

Assign names of up to 8 characters to columns.

NAMEs

(with no parameters)

Display a summary of the names and contents of the columns in the 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.

SAVE a worksheet to a file.

Respond <*filename*> to prompt

TIDY the worksheet

Clean up worksheet for more efficient use.

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.

4 Arithmetic and data editing

ABSOlute values of <*input group*> to <*output group*>

ABSOlute value of <*input value*> {to <*output box*>}

ALOGit **,** antilogit of values in <*input group*> to <*output group*>

ALOGit, antilogit of <*input value*> {to <*output box*>}

ALOGit (*x*) = 1 / (1 + EXPO (-*x*))

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.

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*

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).

CALCulate {<*box*> = } <*expression which may involve boxes and numerals*>

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 the section on the **CALC**ulate command. 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 *also separate section on the **CALC**ulate command*.

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.

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)

See also OMIT.

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

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

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

COUNt the items in <*column*> {, result to <*box*>}

If <*box*> is absent, display only.

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)

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.

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

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*>.

EXPOnential of values in <*input group*> to <*output group*>

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

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}

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)

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.

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)

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).

LOGE logarithm base *e* of values in <*input group*> to <*output group*>

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

LOGIt of values in <*input group*> to <*output group*>

LOGIt of <*input value*> {to <*output box*>}

LOGI(x) = LOGE(x/(1-x))

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

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

MAXimum of corresponding values in columns of <*input group*> results to <*output column*>

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

MINimum of corresponding values in columns of <*input group*> results to <*output column*>

MODulus : modulus operator for example: 5 mod 2 = 1

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.

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*>

PICK rows whose numbers are in <*key column*> from <*input group*> and store values in <*output group*>

Note that it is the positions of the items or rows which are specified. Contrast CHOOse. See also DISCard.

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)

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.

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)

ROUNd values in <*input group*> to nearest integer, results to <*output group*>

ROUNd <*input value*> {to <*output 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.

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 $x > 0$.

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

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

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.

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)

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

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

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

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.

5 Elementary statistical operations

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.

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.

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.

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.

CPRObability, tail probability for value <*value*> from the Chi-squared distribution with <*value*> degrees of freedom, {result to <*box*>}

CPRObability, tail probabilities for values in <*input column*> from the chi-squared distribution with <*value*> | <*column*> degrees of freedom, results to <*output column*>

DUMMy variables from code values in <*category column*> results to <*dummy variable group*>

<*category column*> can contain any values: these values rounded to the nearest integer are considered as codes. The number of columns in <*dummy variable 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).

FPRObability, tail probability for value <*value*> from the F distribution with degrees of freedom <*value*> (numerator) and <*value*> (denominator), {result to <*box*>}

FPRObability, 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*>

GPRObability, tail probability for value <*value*> from the Gamma distribution with shape parameter <*value*> and unit scale parameter {result to <*box*>}

GPRObability, 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*>

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)

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.

NEDeviate, the N(0,1) distribution value corresponding to the cumulative probability <*value*>, {result to <*box*>}

NEDeviates, 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)

NPRObability, tail probability for value <*value*> from the standard Normal distribution, {result to <*box*>}

NPRObability, tail probabilities for values in <*input column*> from the standard Normal distribution, results to <*output column*>

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)

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.

TABStore, tabulate and store counts in <*output column*> using screen-output key <*value*> for a table with categories defined by the values in <*column*>

TABStore, 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*>}

TABStore, 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*>}

TABStore, 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*>

TABStore, 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*>}

TABStore, 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.

TABUlate using output key <*value*> for a table with categories defined by the values in <*column*>

TABUlate using output key <*value*> for a table with categories defined by the values in <*column*> and <*column*> {for values in <*column*> equal to <*value*>}

TABUlate 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*>}

TABUlate the means and s.d.s of values in <*variate column*> for each cell in the table defined

by categories in <*column*>

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*> 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 (signed square root of) 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

TPRObability, tail probability for value <*value*> from the t-distribution with <*value*> degrees of freedom, {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*>

ZSCOres, standardise the values in <*input group*> to have zero mean and unit standard deviation, results to <*output group*>

ZSCO(*x*)=(*x*-mean(*x*))/s.d.(*x*)

6 Model-related data manipulation

EXCLude mode N [C]

EXCL 1 C : Exclude cases from the analysis where corresponding elements in C are non-zero. The length of C should be equal to the number of level 1 units.

EXCL 0 turn exclusion off

If you graphically specify that particular units at any level are to be left out of the analysis via the graph options screen then corresponding elements in the exclude column are set to have non-zero values, other element are left unchanged. If you specify some units to be excluded via graph highlighting and no exclusion column has been given then the software automatically chooses the last free column on the worksheet.

The exclusion column can also be set on the screen that appears when pressing the options button on the hierarchy viewer.

LEV1: merge to level 1

The lev1 operator is useful as a quick way to merge things (e.g. 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.

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).

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)

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.

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)

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.

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)

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)

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)

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.

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.

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)

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)

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.

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).

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.

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*.

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)

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.

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*>}

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*>

7 Multilevel model estimation and control

BATCh {<*value*>}

If <*value*> = 0, turn batch mode off.

If <*value*> = 1, turn batch mode on.

If <*value*> is absent, reverse the current mode.

CLEAr all model specifications

The model specifications and the residual settings are cleared.

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.

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.

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 C(3), 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 C(3). Thus the random part at level 2 now contains only C1 and C(3).

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.

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*>.

FIXEd

Display the current estimates of the fixed parameters

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.

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 standard errors based on raw residuals.

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 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 1 -1 0 0). Note that the 5th parameter, although not part of the contrasts, still occupies spaces in <*contrasts column*>.

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*>.

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

LIKElihood {to <*box*>}

Display the value -2 * LOGE (likelihood for current model) and optionally store in <*box*>.

MAXIterations <*value*> before program halts in batch mode.

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

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 C(1), s.e. residual starting values) in C(2)} {priors in C(3)} 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 (C(1)), 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 opt to supply residual starting values then you must stack residuals for *every* random variable at level 2 and above into the input column.
C(2) is required if C(1) is specified. This column should contain the estimated covariance matrices of the residuals. These should be in the same format as the 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 C(1).

The user also has the option is to supply prior information (C(3)) 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 prior column (C(3)) 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

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 and 3 for Poisson 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 C(1), likelihood to C(2), mean of all updates to C(3), 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 C(1), 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.

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.

NEXT

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

OFFSet at level <*value*> from <*offset column*>

If <*offset column*> contains the vector *v* then *vv'* is subtracted from the residual cross product 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.

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:

OLSEstimates 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*>}

OLSEstimates 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*>}

PREDicted values from fixed part of current model to <*output column*>

PREDicted 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.

Note that where a residual is used that has a *missing value* it is treated as having a value of zero. See the command **Missing residual value**.

RANDom

Display the current estimates of the random parameters.

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 constraints 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*>.

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.

RESPonse variable is in <*response variable column*>

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

RSDErrors type <*value*>

Set the type of standard error computation for random parameters.

If <*value*> = 0, use standard, uncorrected, IGLS or RIGLS computation.

If <*value*> = 2, compute robust standard errors based on raw residuals.

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 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.

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 victories 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.

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.

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.

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.

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*>

See section on cross classifications for an explanation of cross-classified models.

STARt

Start the estimation of the model currently specified.

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.

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.

VFUN level N to <C> [C]

Calculate variance function at level N result to C [se of variance to C]

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 HELP item on weighting for further details.

The following two versions of the WEIGhts command should be used to complete the specification.

WEIGhts 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

WEIGhts

Creates standardised weights (omit if raw weights to be used)

8 Residual estimation

All but two 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 specifications, 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.

MISResiduals <value>. Residual missing 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 (set to missing if within δ of zero)

<value> = 0 Turn off residual missing command

RCOVariances, 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. See RFUNction.

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.

REFLate C..C C..C

Reflate residuals in columns C..C reflated residuals output to C..C

Reflation is based on current estimate of Ω_l. The level is obtained by *MLwiN* using the length

of the residual column

RWEIght mode N

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

RESIduals

Calculate and output residuals according to current RSETttings. See RSETtings, 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.

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.

RLEVel <*value*>

Specify the level at which residuals are to be calculated.

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.

RSETtings

Display residual estimation settings.

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.

9 Macros

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.

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.***

The first set of commands are general macro commands and these are followed by commands for graphics, macro commands which create input screens and macro commands for updating the *MLwiN* front end.

ABORt macro execution and return to terminal input

ASSIgn numbers to <*group*>

This command provides an equivalent function to that of INPUt, but without the necessity for user intervention during execution. 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.

BREAk

Exit from loop. See LOOP.

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

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.

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.

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.

EVARiables, output the column numbers of all explanatory variables to <*column*> {with fitting 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.

EXISt, put indicator of the existence of column <*name*> in <*box*>

<*box*> is set to 1 if column <*name*> exists, otherwise set to 0.

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*\nonlin

causes *MLwiN* to look for macro files in the current directory and then in c:*MLwiN*\nonlin.

FPATh

(with no parameters)

Clear the current search path for macro files.

FSET

Display current FPATh, PREFile and POSTfile settings.

GDIVide <*group*> by <*column*>

Divide each column in <*group*> by <*column*>, results to <*group*> overwriting original

contents. This command has no storage overhead.

GMULtiply <*group*> by <*column*>

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

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

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

IDCOlumn for level <*value*> to <*box*>

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

IDCOlumn <*column*>

Output 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)

IMACro mode <*value*>

If <*value*> = 1, allow interruption of the execution of any macro by pressing the ESCAPE key. In this mode, pressing the ESCAPE key during macro execution has the same effect as executing a PAUSe command. Macro execution can then be resumed by typing RESUme.

If <*value*> = 0, disallow interruption of macro execution.

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.

ITNUmber **0** {<*box*>}

Display the current iteration number and store in <*box*> if specified.

ITNUmber **1** <*value*>

Set the current iteration number to <*value*>.

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.

NFIXed, output the number of fixed parameters to <*box*>

NLEVel, output the number of levels in the current model to <*box*>

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

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

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.

POSTfile <*filename*>

Set macro file to be obeyed after STARt, NEXT, RESIduals or LIKElihood.

POSTfile mode <*value*>

If <*value*> = 0 disable postfile.

If <*value*> = 1 enable postfile.

PREFile <*filename*>

Set macro file to be obeyed before STARt, NEXT, RESIduals or LIKElihood.

PREFile mode <*value*>

If <*value*> = 0 disable prefile.

If <*value*> = 1 enable prefile.

RESUme

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

RETUrn control from current macro file to caller

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*>

RPARameters, store details of the 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.

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.

SAY <*text*>

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.

SUPPress arithmetic warnings

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.

WBUTton S

Creates a button with text string S printed on it.

The following set of commands create a window that contains a button labelled do work, that calls 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
```

YVARiable, output the column number of the response variable to <*box*>

9.1 Macro commands for graphics

The following 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.

GALL C

Change highlight filter column to C.

Sets highlight filter column. C should be length of current data set. 0 leave out from graphs, 1 – plot in default style, 2-17 plot in highlight styles 1-16.

The graph highlight column can also be changed from the graph highlight window.

Note that it is often useful to form arbitrary complex criteria using the CALCulate command for excluding cases from the model or for graphical highlighting(rather than selection by pointing and clicking on graphs). In this case users can assign to the exclude & graphical highlight columns directly.

GBAR N

Set barwidth for current data set of current graph, only applies when data is plotted as a histogram

GCLEar clear all graph sets.

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

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.

GCOLumn for level <value> to <box>

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

GCOOrdinate <value 1> <value 2>

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).

GETYpe <value>

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

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.

GGRId <value>

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

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.

GHIGh mode N

Mode = 0 exclude current data set from highlighting

Mode =1 include current data set from highlighting

Note : data sets included by default

GINDex

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

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)

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.

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.

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.

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

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.

GSSZ <value>

Set the symbol size for the current data set, measured in thousandths 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.

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

GTEXt mode N

Show text labels assigned to data sets as :

N=0 hide all text,

N=1 in a single legend regardless of how many graphs in the display

N=2 in legends with one legend per graph

N=3 in labels

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”

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>)’the 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”

GTYPe <value>

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

GTYPe <value>

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

GXCOl <column>

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

GYCOl <column>

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

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.
GYERror 2 : clear negative errors

9.2 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.

IMAC **<value 1>**

As described above, 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 pressing 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** 1or **PAUSE 2** 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.

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.

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 MCREs will then calculate residuals based on the current MCMC model.

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.

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.

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

OBPAth **S1**

Place the current fpath setting into string variable S1

PAUSE 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 an error occurs during execution of a macro then, when PAUSE 1 is present the macro is paused until the ENTER key is pressed.

PAUSE 2

This has the same function as PAUSE 1, but does not pause the macro if an error occurs. Instead, the error message is written to the OUTPUT window where it can be viewed. This is useful, fro example in simulations, where the user wishes the macro to proceed even where errors are flagged.

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.

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 C96-C99 accordingly. The values that were in C96 and C98 are then treated as the previous estimates.

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.

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".

STRIngs **displays contents of the 20 string variables.**

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.

.

10 Simulation

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 from C1 and outputs the results to C2.

Combined with the LINK and SORT commands the BOOTstrap command can be used to carry out bootstrap simulations of models. For example,

```
LINK -4 G1          {record in G1 all columns referenced by the model}
CALC G2=G1          {take a copy}
LOOP B1 1 50        {take 50 bootstrap samples}
 BOOT 5000 G2 G1    {bootstrap a sample of 5000 into G1}
 SORT 3 G1 G1       {sort on unit IDs - assume a 3-level model here}
 STAR               {analyse this sample}
 JOIN C56 C96 C56   {stack up random parameter estimates}
 JOIN C58 C98 C58   {stack up fixed parameter estimates}
ENDL
```

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 from C1 and outputs the results to C2.

Combined with the LINK and SORT commands the BOOTstrap command can be used to carry out bootstrap simulations of models. For example,

```
LINK -4 G1          {record in G1 all columns referenced by the model}
CALC G2=G1          {take a copy}
LOOP B1 1 50        {take 50 bootstrap samples}
 BOOT 5000 G2 G1    {bootstrap a sample of 5000 into G1}
 SORT 3 G1 G1       {sort on unit IDs - assume a 3-level model here}
 STAR               {analyse this sample}
 JOIN C56 C96 C56   {stack up random parameter estimates}
 JOIN C58 C98 C58   {stack up fixed parameter estimates}
ENDL
```

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.*

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.*

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

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

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

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

SEED, reseed the random number generator using the start value <*value*>

SEED, reseed the random number generator using the start value <*value*>

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)’

SIMUlate <*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 SIMU command useful when residuals are not all multivariate Normal, for example in Binomial models, is:

SIMUlate {<*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*>.

To simulate 1129 responses for a 3-level binomial model and place them in C103:

```
SIMU  2  3  C100
PRED  C101
CALC  C102 = ALOG (C100+C101)
BRAN  1129  C103  C102  'DENOM'
```

TLRAandom cutpoints in C, below(0)/above(1) in C, output to C

Sample from truncated logistic

TNRAndom cutpoints in C, below(0)/above(1) in C, output to C

Sample from truncated normal

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

11 Matrices

BLKTotals, for block number <*value*>, place the total 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.

CHOLeskyk C C

Takes symmetric matrix in shortened form in the first column and outputs its cholesqy decomposition in square matrix form thus

->join 1 0.5 2 c1

->chol c1 c2

->prin c1 c2

```
      C1          C2
 N =     3           4
   1  1.0000       1.0000
   2  0.50000      0.50000
   3  2.0000       0.00000
   4                1.3229
```

->calc c3=c2*.(~c2)

->prin c3

```
      C3
 N =     4
   1  1.0000
   2  0.50000
   3  0.50000
   4  2.0000
```

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

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

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.

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.

MVIEw with matrix identifier(s) in <*ID column*> the stacked lower-triangular matrices in <*matrix column*>

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),

MVIEw C1 C2 will produce the display

```
1

2   3

4   5   6


7

8   9

10  11  12
```

together with row and column headings.

Thus, different codes in <*ID column*> identify different matrices, and there are as many of a given code as there are rows in the corresponding lower-triangular matrix.

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*>

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*>.

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*>.

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*>

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**.

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

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

ZMATrix, output Z, the random-part design matrix corresponding to unit number <*block value*> for explanatory variables random at level <*level value*> to <*output column*>

<*block value*> corresponds to a unit at the highest level. <*level value*> determines the explanatory variables which are to be output, and <*block value*> determines the cases for which values on these variables are to be output.

11.1.1 Some examples

```
Assign c1-c3
1 2 3
4 5 6
7 8 9
10 11 12
link c1-c3 g1
```

We can form the transpose (~) of g1

```
calc g2=~g1
prin g2
       C400         C399         C398         C397
N =       3            3            3            3
  1  1.0000       4.0000       7.0000       10.000
  2  2.0000       5.0000       8.0000       11.000
  3  3.0000       6.0000       9.0000       12.000
```

The columns chosen to store the result (g2) are the highest unused columns on the worksheet. If the result group already contains the right number of columns then these columns are used.

We can form the matrix product (*.) of g1 with its transpose:

```
calc g2=g1*.(~g1)
prin g2
      C400         C399         C398         C397
N =       4            4            4            4
  1  14.000       32.000       50.000       68.000
  2  32.000       77.000       122.00       167.00
  3  50.000       122.00       194.00       266.00
  4  68.000       167.00       266.00       365.00
```

If one of the operands is a matrix then the operators +, /, -,* are defined as outer operations on the matrix. For example,

```
calc g2=g1+g1
prin g1
         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.

```
assign 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
```

11.1.2 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 MATRix 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:

```
assign 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
```

11.1.3 Summary

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

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

12 Miscellaneous commands

COMMand editing {<*value*>}

If <*value*> = 0 turn command editing off.

If <*value*> = 1 turn command editing on (the default setting).

If <*value*> is absent, alternate between on and off.

When command editing is on, previous commands can be retrieved and any displayed command modified with the help of the arrow keys.

ECHO {<*value*>}

If <*value*> = 0 turn echoing off (default setting)

If <*value*> = 1 turn echoing on.

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.

LOGAppend

Respond <*filename*> to prompt.

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

LOGOn or off {If logging on type <*filename*>} {If logging off type <*0*>}

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.

Note that several log files can be created by successively logging on and off.

NOTE <*text*>

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

STOP

Respond **Y** or **N** to prompt.

Quit *MLwiN*. You will be asked whether you wish to save the current worksheet.

WAIT {<*value*>}

If <*value*> = 0 turn waiting off.

If <*value*> = 1 turn waiting on (the default setting).

If <*value*> is absent, alternate between on and off.

If waiting is on, *MLwiN* pauses when certain things happen, inviting the user to respond when ready. For example, when the displayed output from a PRINt command fills the screen the user must press RETURN to view further output; when a SAVE command would over-write an existing file the user is warned and invited to confirm that this is wanted; when data are input from or output to a file, the user is prompted for a filename and possibly also a format. For interactive use, waiting is normally on.

When commands are executed in macros, however, it can be inconvenient for the user to have to respond repeatedly to such requests. In this case waiting should be turned off.

13 The CALCulate command

The format of the CALCulate command is one of:

CALCulate {<*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.

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

Operators, if present, can be arithmetic, relational, logical, or matrix. The arithmetic operators are:

The functions ABSOlute, ALOGit, ANGUlar, ANTIlogarithm, COSine, DIVide, EXPOnential, LOGE, LOGIt, LOGTen, MODulus, NEDeviate, ROUNd, SIGN, SINe, SQRT, 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

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

The logical operators are:

&	and
|	or
!	not

These operate on the results *true* and *false* in the obvious way.

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

Finally, there are the following matrix operators:

These are explained in Appendix B.

level	operators
1 (low)	&, |
2	<, >, >=, <=, !=, ==
3	+, -
4	*, /
5	*.
6	unary -, ~, !
7 (high)	named functions NED, INV etc.

The order of precedence of the operators is:

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.

13.1 Examples

The CALCulate command treats groups as matrices – see section on 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 - see section on 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
   5  3.0000       18.000       15.000
```

Here the expressions in parentheses are evaluated first, to produce a set of 4 zeroes (*false*) and a 1 (*true*). The expression 2 * c4 is then evaluated, to produce the 5 values 2, 4, 6, 8, 10. The contents of C4 are not changed. These 5 values are then multiplied one by one by the 4 zeroes and a 1 already produced. The result is added to the original contents of C4 and finally assigned to C4.

14 Access to the multilevel algorithm

A number of commands give access to the matrices used by the estimation algorithm of *MLwiN*. Their formats are fully described in the section Matrix Commands. Here we give some examples of their use, followed by annotated listings of a set of macros which imitate *MLwiN*'s estimation of model parameters.

14.1 Examples

The example here uses a data set called JSPMAT.ws. A copy can be obtained from the *MLwiN* web site.

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
       C400         C399         C398         C397         C396
N =      5            5            5            5            5
  1  28.678        1.8874       1.2935       1.7885       3.0752
  2  1.8874        30.032       1.4084       2.6060       5.7198
  3  1.2935        1.4084       28.423       1.3797       1.7530
  4  1.7885        2.6060       1.3797       29.588       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
```

ZMATrix, output Z, the random-part design matrix corresponding to unit number <*block value*> for explanatory variables random at level <*level value*> to <*output column*>

<*block value*> corresponds to a unit at the highest level. <*level value*> determines the explanatory variables which are to be output, and <*block value*> determines the cases for which values on these variables are to be output.

For example

```
zmat 42 2 c100
calc g1=c100
```

```
prin g1
        C389          C388
 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
```

Other useful commands are:

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

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*>

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*>

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

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

14.2 A macro implementation of the estimation algorithm

The macros in this section are contained in the file ALGOR of the directory where *MLwiN* is stored.

The files are:

RUN

ITER

RAND

NAMES

DOXXR

FIXED

KRON

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 using the commands on the SETTings screen or via the GUI. 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.

14.2.1 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 ITNUmber 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 achieved]
  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.

14.2.2 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
```

14.2.3 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
```

14.2.4 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			
		$B_1 = x_0 x_0^T$	$B_2 = x_0 x_1^T + x_1 x_0^T$	$B_3 = x_0 x_0^T$	$B_4 = diag(x_0 x_0^T)$
A	$A_1 = x_0 x_0^T$	$tr(A_1 V^{-1} B_1 V^{-1})$			
	$A_2 = x_0 x_1^T + x_1 x_0^T$	$tr(A_2 V^{-1} B_1 V^{-1})$	$tr(A_2 V^{-1} B_2 V^{-1})$		
	$A_3 = x_0 x_0^T$	$tr(A_3 V^{-1} B_1 V^{-1})$	$tr(A_3 V^{-1} B_2 V^{-1})$	$tr(A_3 V^{-1} B_3 V^{-1})$	
	$A_4 = diag(x_0 x_0^T)$	$tr(A_4 V^{-1} B_1 V^{-1})$	$tr(A_4 V^{-1} B_2 V^{-1})$	$tr(A_4 V^{-1} B_3 V^{-1})$	$tr(A_4 V^{-1} B_4 V^{-1})$

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.

macro KRON commands

```
set b61 1                    [b61 counts through elements of xxr]
calc 'yy'= 'yr' *. (~'yr')   [form Y** for current block]
loop b60 1 b54               [b60 : 1->r]
  obey doxxr                 [calculate ith row of xxr]
  calc 'temp'= diag(g1[b60]*.'vinv'*.'yy'*.'vinv')      [diag(A_i V^-1 Y V^-1)]
  sum 'temp' b62             [get trace..]
  pick b60 'xyr' b71         [..and accumulate]
  calc b71=b71+b62
  edit b60 'xyr' b71
endloop
```

14.2.5 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
```

14.2.6 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]
```

To run the 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.

```
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.

15 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.

15.1 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 and Goldstein (1994).

15.2 Some computational considerations

Cross-classified models can demand large amounts of storage for their estimation and can be computationally intensive. The storage required to run a model, in worksheet cells, is given by:

$$3n_e b + n_f b + \sum_{\ell=1}^{\ell=L} 4r_\ell b + 2br_{\max} \qquad (5)$$

where

b is the number of level-1 units in the largest highest-level unit

n_e is the number of explanatory variables

n_f is the number of fixed parameters

L is the number of levels in the model

r_ℓ is the number of variances being estimated at level ℓ (covariances add no further space requirements)

$r_{\max}$ is the maximum number of variances at a single level

In cross-classified models n_e will be large, typically the size of the smaller classification, and *b* will be equal to the size of the entire data set. These facts can lead to some quite devastating storage requirements for cross-classified models. In the example of 100 schools crossed with 30 neighbourhoods, suppose we have data on 3000 pupils. The storage required to handle the computation for a cross-classified variance components model is:

$$3n_e b \quad + n_f b \quad + \sum_{l=1}^{l=L} 4r_l b \quad + 2br_{\max}$$

$$3*30*3000+3000 \quad + 4(3000+3000+30*3000) \quad + 2*3000*30$$

that is, some 840,000 free worksheet cells. If we wish to model a slope varying across neighbourhoods then the storage requirement doubles.

These storage requirements can be reduced if we can split the data into separate groups of schools and neighbourhoods. For example, if 40 of the 100 schools take children from only 12 of the 30 neighbourhoods, and no child from those 12 neighbourhoods goes to a different school, then we can treat those 40 schools by 12 neighbourhoods as a separate group. Suppose that in this way we can split our data set of 100 schools and 30 neighbourhoods into 3 separate groups, where the first group contains 40 schools and 12 neighbourhoods, the second contains 35 schools and 11 neighbourhoods and the third contains 25 schools and 7 neighbourhoods. We can then sort the data on school within group and make group the third level. We can do this because there is no link between the groups and all the covariances we wish to model are contained within the block diagonal matrix defined by the group blocks.

For a cross-classified variance components model n_e is now the maximum number of neighbourhoods in a group, that is 12, and *b* is the 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 command XSEArch is designed to find such groups in the data.

15.3 An example of a 2-way cross classification

In this example we analyse children's overall exam attainment at age sixteen. 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.

The initial analysis requires a worksheet size of 850,000 cells. The default size for *MLwiN* is 250,000: to run with the larger worksheet use the INIT command to allow this number of cells.

If you do not have enough memory on your machine to allocate such a large worksheet you can read through the rest of this section and resume modelling in the next section.

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

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.

15.3.1 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.

15.4 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.

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)

The expression given earlier describes the amount of memory *MLwiN* requires to run any given model. The two key terms are b, the number of level-1 units in the largest highest-level unit, and $r_{\max}$, the maximum number of random parameters at a single level. With cross-classified models b corresponds to the number of level-1 units in the largest separated group and $r_{\max}$ corresponds to the maximum number of non-hierarchical units in any single separated group.

In the example in the previous section we used XOMIt to delete cells with 2 or fewer observations: this resulted in the loss of 168 observations in total. XSEArch then found 6 separate groups with a maximum of 8 secondary schools in a group ($r_{\max}$ = 8). If we look at the number of pupils in each group by typing

```
tabu 1 c13
                  1          2          3          4          5

       N        973       1330        460        320        106
       %       29.8       40.7       14.1        9.8        3.2
                  6     TOTALS

       N         78       3267
       %        2.4      100.0
```

we see that group 2 contains the largest number of level 1 units (b=1330). For this procedure $br_{\max}$ = 1330*8 = 10640, at the cost of losing 168 observations.

15.5 A multi-way 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   {take a run of random numbers before starting simulation}
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.

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*>}

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.

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.

16 Multiple membership models

There are two new commands which together can be used for specifying multiple membership models. These are **WTCO**l 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

WTCOl 4, id columns C1-C4, weights in C5-C8, weighted indicator columns output to C101-C200

Suppose pupil 1 attends only school 5, pupil 2 attends schools 8 and 9 with proportions 0.4 and 0.6, pupil 3 attends schools 4,5,8,6 with proportions 0.2,0.4,0.3,0.1; then the id and weight columns for these 3 pupils would contain the data

c1	c2	c3	c4	c5	c6	c7	c8
5	0	0	0	1	0	0	0
8	9	0	0	0.4	0.6	0	0
4	5	8	6	0.2	0.4	0.3	0.1

The first 9 columns of the output for these three children would be

c101	c102	c103	c104	c105	c106	c107	c108	c109
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0.4	0.6
0	0	0	0.2	0.4	0.1	0	0.3	0

The second command is the ADDM command.

This command adds sets of weighted indicator variables created by the WTCOL command to the random part of the model at level M and generates a constraint matrix which defines the variances estimated for each set of indicators to be equal. It is possible to have more than one set of indicators if you wish to allow several random coefficients to vary across the multiple membership classification.

First consider, a variance components multiple membership model, continuing from the example outlined in the description of the WTCOl command; in this case we need :

ADDM 1 set of indicators at level 2, in C101-C200, constraints to C20

If we wished to allow the slope of an X variable, say “PRETEST”, in addition to the intercept, to vary randomly across the multiple membership classification, we must then form another set of indicators which are the product of the original indicators and “PRETEST”.

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.

WTCOl

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

ADDM

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

17 References

Rasbash, J., Browne, W., Goldstein, H., Yang, M., et al. (2000). *A user's guide to MLwiN (Second Edition).* London, Institute of Education:

Yang, M., Rasbash, J., Goldstein, H. and Barbosa, M. (2000). *MLwiN Macros for advanced multilevel modelling.* London, Institute of Education:

Hill, P. W. and Goldstein, H. (1998). Multilevel modelling of educational data with cross classification and missing identification of units. *Journal of Educational and Behavioural statistics* **23**: 117-128.

Rasbash, J. and Goldstein, H. (1994). Efficient analysis of mixed hierarchical and cross classified random structures using a multilevel model. *Journal of Educational and Behavioural statistics* **19**: 337-50.

Goldstein, H. and Rasbash, J. (1992). Efficient computational procedures for the estimation of parameters in multilevel models based on iterative generalised least squares. *Computational Statistics and Data Analysis* **13**: 63-71.

18 INDEX